Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().

Rebu, estensione al problema

Con riferimento al caso di studio REBU si consideri un nuovo requisito. Per motivi di sicurezza, sono state introdotte delle limitazioni al numero di ore consecutive di lavoro per un autista, e definiti dei tempi di riposo prima del turno successivo. In particolare, un turno di lavoro può durare al massimo 9 ore, di cui una dedicata a una pausa pranzo, da effettuarsi non prima di 3 e non dopo 5 ore di guida. Eventuali altre pause non verranno conteggiate come periodo di recupero. Tra un turno e il successivo devono passare almeno 12 ore. Infine, dopo un turno notturno (un turno si intende come notturno se include almeno 5 ore di lavoro fra le 22:00 e le 6:00) sono richieste 24 ore di riposo. Quando un autista chiede di iniziare un nuovo turno, il sistema controlla che i tempi di riposo siano stati rispettati, quindi monitora l'autista durante il turno, imponendo uno stop ove necessario. L'autista segnala inizio e fine turno e inizio e fine di ogni pausa, se non già imposte dal sistema. Il sistema non "passa" prenotazioni a un autista che non abbia rispettato le limitazioni sui turni.

Caso d'uso AutorizzaTurno

- Si consideri il caso d'uso AutorizzaTurno, avente la seguente SPE:

1. L'autista chiede di poter iniziare un turno
2. Il sistema recupera i dati dell'autista
3. Il sistema verifica il rispetto delle condizioni
4. SE verificato, il sistema autorizza;
5. ALTRIMENTI il sistema rifiuta.

Architettura

- La progettazione architetturale ha individuato le seguenti componenti:
  - **DB autisti,** che mantiene informazioni sugli autisti, comprensive del loro conto economico;
  - **Gestione Autisti,** che realizza la business logic.
  - **App Autista,** che realizza l'interfaccia dell'autista
- Dare
  - la vista C&C
  - un diagramma di sequenza che mostri come queste componenti e gli attori coinvolti realizzano il caso d'uso AutorizzaTurno.
  - Una vista di dislocazione

SOLID

- Dare la definizione dei principi di progettazione SOLID, commentando cosa si intende con Single Responsibility Principle ed esemplificare il principio applicandolo al caso di studio.

La classe *Turno*

Abbiamo deciso di rappresentare il turno di lavoro di un autista con una classe *Turno*, contenente fra l'altro una sequenza di *Segmento*, ciascuno dei quali ha fra i suoi attributi di un'ora di *inizio* e di un'ora di *fine* di un periodo di lavoro consecutivo (gli intervalli fra due segmenti consecutivi sono considerati di pausa).
La classe potrebbe avere un'implementazione simile alla seguente (leggermente semplificata):

```
public class Turno {
	private Vector<Segmento> segs = new Vector<Segmento>();
	private Segmento cur = null;

	public int size() { return segs.size(); }
	public Segmento get(int i) { return segs.get(i); }
	public void begin() { cur = new Segment( now(), -1 ); }
	public void end() { cur.fine = now(); segs.add(cur); cur=null; }
	public boolean status() { return cur!=null ? WORKING:PAUSE; }
	public boolean isLegal() { … }
	public boolean isNightShift() { … }
	private long now() { return System.currentTimeMillis(); }
}
```

- La progettazione prevede che quando un autista inizia un turno venga creato un nuovo oggetto *t* (il "turno corrente" dell'autista)
- Inoltre all'inizio di un periodo di lavoro (inizio turno o dopo una pausa) viene invocato t.begin() e alla fine del periodo t.end().
- In ogni momento, si possono invocare t.size() e t.get() con l'ovvia semantica, mentre t.status() indica se in un dato momento l'autista è in pausa o sta lavorando.
- Il metodo t.isNightShift() restituisce true se *t* può essere classificato come turno notturno, mentre t.isLegal() è inteso restituire true se *t* rappresenta un turno di lavoro che rispetta le regole di sicurezza date nel testo.
- Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal().

Progettazione di dettaglio

- Dare il diagramma di struttura composita di **Gestione Autisti** (che comprenda parti di tipo Turno)

Verifica

Con le convenzioni precedenti,

1. Si scriva un test di unità che si proponga di verificare il corretto funzionamento dei metodi size(), get(), begin(), end().

2. Per semplicità, si consideri `isLegal2(Vector<Segmento> segs)` e dice se `segs` è legale
3. Si definisca un insieme di casi di test per il metodo isLegal2(), usando il criterio della partizione dei dati di ingresso in classi di equivalenza, e quello della verifica dei casi di confine. Opzionalmente, si faccia una stima sul grado di copertura della test suite data.
   Ovviamente, su un singolo turno non si può verificare se il periodo di riposo dopo un turno notturno sia stato di almeno 24 ore -- questa condizione quindi non farà parte di quelle controllate da isLegal2().