Tercer Control AS – Grup 10 **4-6-18**

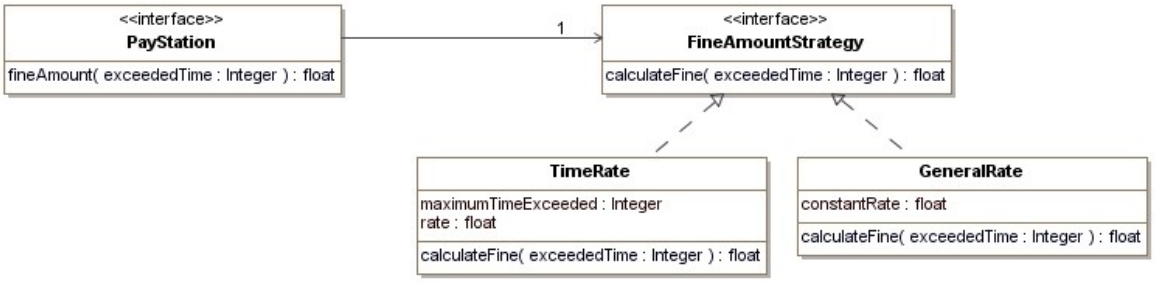
Nom: .....

Ens han demanat afegir una nova història d'usuari al software PayStation (que hem estat desenvolupat a la classe de laboratori) per tal de que els usuaris dels parquímetres puguin consultar l'import de les multes per sobrepassar el temps d'estacionament pagat.
Quan un conductor arribi al vehicle i trobi una multa es podrà dirigir al parquímetre per conèixer l'import de la multa que haurà de pagar. El conductor haurà d'indicar en el parquímetre el temps excedit en minuts (indicat a la multa trobada al vehicle) i el parquímetre informarà de l'import de la multa a pagar. Inicialment, el software oferirà dues maneres diferents per calcular l'import de les multes:

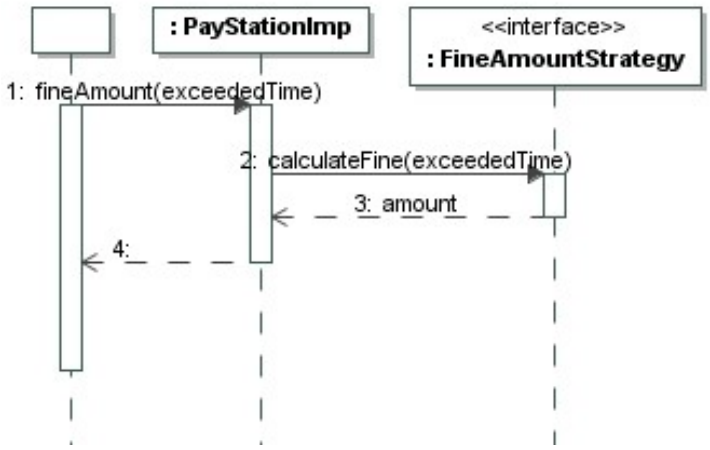
1) *Tarifa general*. L'import de la multa serà un valor constant (enter) i fixat que s'haurà inicialitzat en donar d'alta el parquímetre. Aquest valor pot ser diferent per cada parquímetre i pot ser canviat quan es vulgui pels administradors del parquímetre.
2) *Tarifa en funció del temps*. L'import de la multa dependrà de dos valors fixats que s'hauran d'inicialitzar en donar d'alta el parquímetre, *tempsExceditMàxim* i *tarifa* (enters). Si el temps excedit és superior al *tempsExceditMàxim* s'activarà l'excepció *NoEsPotCalcularMulta*. En cas contrari, l'import de la multa serà el temps excedit multiplicat per la *tarifa*.

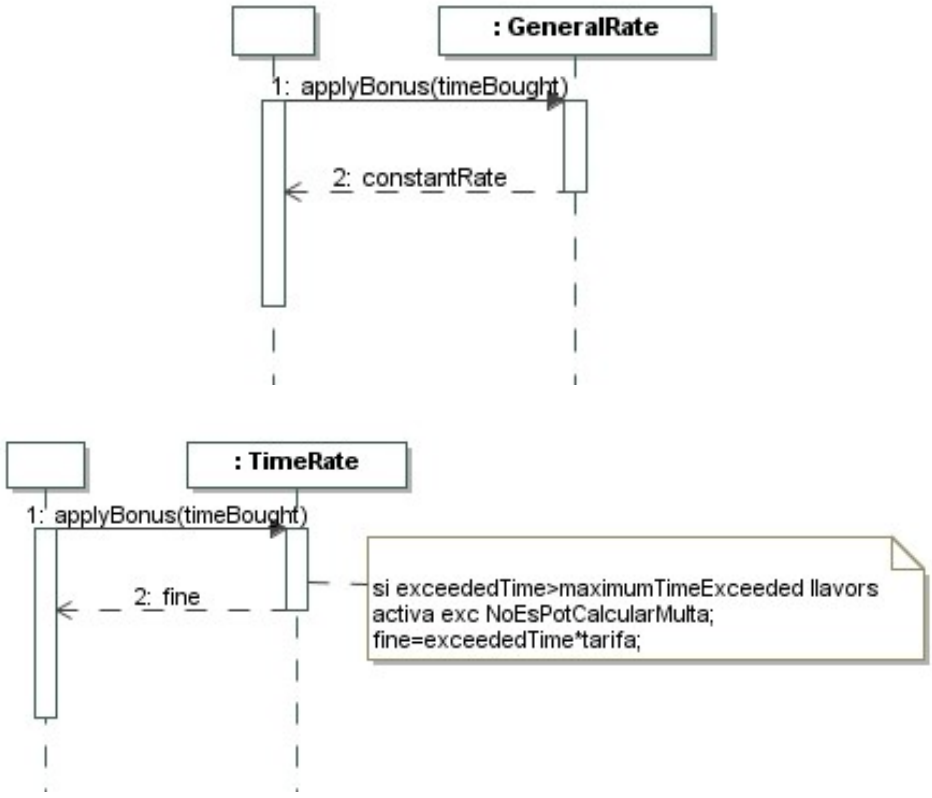
Es demana:
1. [3 punts] (Temps estimat: 15 minuts) (Competència transversal) Expliqueu de forma **detallada** i **raonada** la solució que proposaríeu per afegir aquesta història d'usuari tant des del punt de vista estàtic (**diagrama de classes**) com des del punt de vista del comportament (**diagrama de seqüència**). Volem una solució que: 1) permeti als parquímetres seleccionar una forma de càlcul de l'import de les multes a l'inici i canviar-la en temps d'execució i 2) permeti afegir noves formes de calcular l'import de les multes (sense fer canvis, només extensions). No cal fer servir el patró factoria. **Contesteu aquesta pregunta a l'espai que teniu disponible.**

Utilitzaríem una solució on aplicaríem el patró estratègia. El comportament variable del càlcul de l'import de les multes quedaria encapsulada en una interface i les classes que implementarien aquesta interface implementarien el càlcul concret que han de tenir els parquímetres. A continuació es mostra el diagrama de classes per aquesta solució:



A la figura següent es mostra el diagrama de seqüència per aquesta solució:





2. [3 punts] (Temps estimat: 20 minuts) Expliqueu clarament les iteracions que seguireu per implementar aquesta nova història d'usuari al software PayStation utilitzant TDD. Cal que quedi clar el que fareu a cada iteració. **Contesteu aquesta pregunta a l'espai que teniu disponible.**

- Iteració 1: l'operació no existia i per tant cal començar amb una iteració per definir un test sobre aquesta operació.
  - Definir test per calcular l'import d'una multa amb un fake.
  - Definir el codi per tal de que passi el test.
- Iteració 2 (Refactoring):
  - Introduïr la interface FineAmountStrategy
  - Refactoritzar PayStationImpl per introduïr la referència a la interface FineAmountStrategy i usar la referencia per fer el càlcul (comprovar que el test corresponent falla ja que la interface no té cap classe que la implementa).
  - Definim un nou test case amb una estratègia SimpleRate (retorna un fake) per provar que el codi de pay station funciona. La implementació de l'estratègia s'ha de definir en la carpeta de test ja que el codi d'aquesta estratègia no estarà dins del codi de producció.
- Iteració 3 (General Rate): Definim un nou test case, TestGeneralRate per definir els tests relacionats amb aquest càlcul.
  - Definim el test per provar aquesta estratègia amb una constantRate de 10.
  - Definim el codi per tal de que el test anterior passi.
- Iteració 4 (TimeRate): Definim un nou test case, TestTimeRate per definir els tests relacionats amb aquest càlcul.
  - Definim el test per provar aquesta estratègia (cas vàlid) amb temps excedit 10, temps màxim 20 i tarifa 5. El resultat ha de ser 50.
  - Definim el codi per tal de que el test anterior passi.
  - Triangulació si cal
- Iteració 5 (TimeRate):
  - Definim el test per provar aquesta estratègia (cas excepció). amb temps excedit 20, temps màxim 10 i tarifa 5. S'espera que s'activi l'exepció.
  - Definim el codi per tal de que el test anterior passi.
- Iteració 6 (Test Integració):

- Definir un test case TestIntegration per provar que els pay stations amb les dues formes de càlcul funcionen tal i com s'espera.

3. [4 punts] (Temps estimat: 60 minuts) Implementeu aquest nou requisit utilitzant TDD seguint les iteracions de l'apartat anterior. **El codi inicial de la iteració 9 el teniu disponible a l'Atenea (a l'apartat del control 3). El codi resultant d'afegir la nova història d'usuari l'haureu d'entregar al mateix apartat de l'Atenea en un fitxer comprimit. Poseu com a nom del fitxer el vostre nom i primer cognom.**
CODI FONT

PayStationImpl

```
public class PayStationImpl implements PayStation {
…
  private FineAmountStrategy strategy;

  public PayStationImpl(FineAmountStrategy strategy) {
        this.strategy = strategy;
  }

  public int fineAmount(int exceededTime) throws FineAmountNotAvailable
  {
        return strategy.calculateFine(exceededTime);
  }
}
```

FineAmountStrategy

```
public interface FineAmountStrategy {

        int calculateFine(int exceededTime) throws FineAmountNotAvailable;

}
```

GeneralRate

```
public class GeneralRate implements FineAmountStrategy{
int constantRate;

        public GeneralRate(int i) {
        this. constantRate = i;
}

        public int calculateFine(int exceededTime) throws FineAmountNotAvailable {
                return (constantRate);
        }

}
```

TimeRate

```
public class TimeRate implements FineAmountStrategy{
int rate;
int maximumExceededTime;
```

```
	public TimeRate(int i, int j) {
	this.rate = i;
	this.maximumExceededTime = j;
}

	@Override
	public int calculateFine(int exceededTime) throws FineAmountNotAvailable{
		if (exceededTime > maximumExceededTime)
			throw new FineAmountNotAvailable();
		return exceededTime*rate;
	}

}
```

TESTS

TestPayStation

```
public class TestPayStation {
  PayStation ps;
  /** Fixture for pay station testing. */
  @Before
  public void setUp() {
    ps = new PayStationImpl(new SimpleRate());
  }


  @Test
  public void shouldCalculate0Iteration1() throws FineAmountNotAvailable {
    assertEquals( "Should return 0", 0, ps.fineAmount(20));

  }
}
```

SimpleRate

```
public class SimpleRate implements FineAmountStrategy{

	@Override
	public int calculateFine(int exceededTime) {
		// TODO Auto-generated method stub
		return 0;
	}

}
```

TestGeneralRate

```
public class TestGeneralRate {

	GeneralRate gr;

	 @Before
	  public void setUp() {
	    gr = new GeneralRate(50);
	  }

	@Test
	  public void shouldCalculate50() throws FineAmountNotAvailable {
	    assertEquals( "Should calculate 50",
			50, gr.calculateFine( 20 ));
	  }

}
```

TestTimeRate

```
public class TestTimeRate {
        TimeRate tr;

         @Before
         public void setUp() {
           tr = new TimeRate(10,5);
         }

        @Test
         public void shouldCalculate40For4minutes() throws FineAmountNotAvailable {
           assertEquals( "Should calculate 40 cents for 4 minutes", 4*10, tr.calculateFine( 4 ));
         }

         @Test(expected=FineAmountNotAvailable.class)
         public void shouldNotCalculateFine() throws FineAmountNotAvailable {
           tr.calculateFine(10);
         }

}
```

TestIntegration

```
public class TestIntegration {
        PayStation ps;


         @Test
         public void shouldCalculate30ForGeneralRate() throws FineAmountNotAvailable {
                ps = new PayStationImpl(new GeneralRate(30));
           assertEquals( "Should calculate 30",
                    30, ps.fineAmount( 60 ));
         }
                @Test
                 public void shouldCalculate50ForTimeRate() throws FineAmountNotAvailable {
                        ps = new PayStationImpl(new TimeRate(5,15));
                   assertEquals( "Should calculate 10 cents for 2 minutes",
                              10*5, ps.fineAmount( 10 ));
                 }


                @Test(expected=FineAmountNotAvailable.class)
                 public void shouldNotCalculate() throws FineAmountNotAvailable {
                                ps = new PayStationImpl(new TimeRate(5,15));
                   ps.fineAmount(20);
                 }

}
```