Segon Control AS **17-12-20**

2. [**6 punts]**

1. **[0,5 punts]** Explica quina és la part del codi de PayStationImpl que variarà entre els diferents països (assumim que les ciutats i països amb la mateixa divisa accepten les mateixes monedes). A continuació disposeu el codi actual del PayStationImpl.

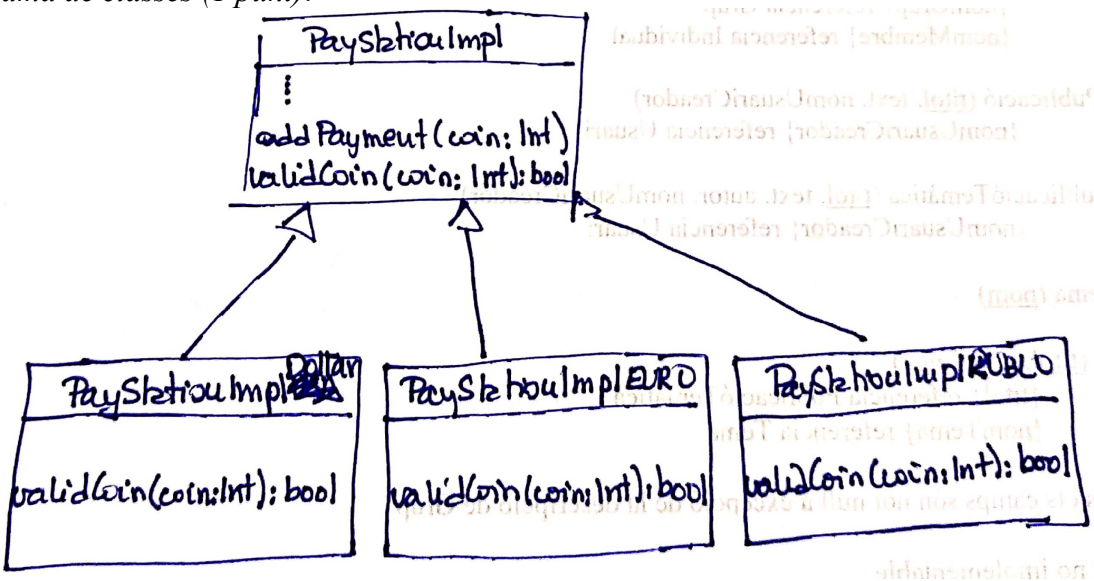
   Resposta: Variarà el que hi ha a dins del switch ja que les monedes acceptades seran diferents en funció del tipus de moneda. No variarà l'activació de l'excepció.

2. **[2,5 punts] Expliqueu** de forma **detallada** i **raonada** quina solució proposaríeu per afegir el nou requisit. Explica també un **avantatge** de la solució que proposeu. Descriviu la vostra solució des del punt de vista estàtic (**diagrama de classes complet amb les operacions**). Volem una solució que permeti afegir en el futur noves divises d'altres països (sense fer canvis, només extensions).

   Resposta:
   Explicació de la solució (1 punt): La solució que aplicaríem seria la solució polimòrfica (aplicació del patró plantilla). La part específica seria el switch (menys l'activació de l'excepció) que variaria per cada divisa. Aquesta part estaria en una operació validCoin(coin:Integer):boolen que retornarà un cert en el cas que la moneda sigui una de les acceptades pel parquímetre i fals en cas contrari. La part comuna seria la resta de l'operació addPayment. La resta de les operacions no es veuen modificades.

   Avantatge (0,5 punts): Compleix el principi d'obert i tancat ja que la introducció de noves divises implica afegir una nova classe i operació però no la modificació de cap operació. Altres avantatges són: 1) evita el problema de manteniment ja que no hi ha codi repetit i 2) no genera operacions amb moltes línies de codi (ifs per cada divisa).

   Diagrama de classes (1 punt):



3. **[3 punts] Expliqueu** clarament les **iteracions** (com estaven definides a les classes de laboratori) que seguireu per implementar aquest canvi al software PayStation utilitzant TDD. Cal que quedi clar el que fareu a cada iteració i els TestCases que definireu.

   Resposta:

   Iteracions (Versió Plantilla):

   Iteració 1: (Refactoring): L'operació addPayment ja existia i estava treballant amb la divisa Dollar.

- A la classe PayStationImpl, extreure un mètode validCoin amb la part del switch que varia (tot el switch menys l'excepció). Aquest mètode ha de retornar cert per a les monedes vàlides i fals en cas contrari.
- A la classe PayStationImpl, caldrà modificar l'operació addPayment per tal de que si el retorn de l'operació és fals, s'activi l'excepció. Una altra opció és definir un stub que implementi aquesta operació.
- Crear una nova classe PayStationImplDOLLAR i moure l'operació validCoin a aquesta classe. A la classe PayStationImpl deixar una implementació de l'operació que retorni true.
- Crear un nou test case TestPayStationImplDOLLAR i moure els tests de TestPayStation (corresponents a les monedes acceptades i rebutjades) a aquest test case.

Iteració 2 (2 cèntims EURO): Definiu un nou test case, TestEURO per definir els tests relacionats amb les monedes vàlides de l'EURO.

- Definir el test per provar una moneda vàlida d'EURO (2 cèntims).
- Definir el codi per tal de que el test anterior passi (incloent la creació de la classe PayStationImplEURO).

Iteració 3 (5 cèntims EURO):

- Definir el test per provar una moneda vàlida d'EURO (5 cèntims).
- Definir el codi per tal de que el test anterior passi.

Iteració 4 (10 cèntims EURO):

- Definir el test per provar una moneda vàlida d'EURO (10 cèntims).
- Definir el codi per tal de que el test anterior passi.

Iteració 5 (20 cèntims EURO):

- Definir el test per provar una moneda vàlida d'EURO (5 cèntims).
- Definir el codi per tal de que el test anterior passi.

Iteració 6 (7 cèntims EURO):

- Definir el test per provar una moneda invàlida d'EURO (7 cèntims).
- Definir el codi per tal de que el test anterior passi (s'activi una excepció).

Iteració 7 (5 kopeks RUBLO): Definiu un nou test case, TestRUBLO per definir els tests relacionats amb les monedes vàlides del RUBLO.

- Definir el test per provar una moneda vàlida de RUBLO (5 kopeks).
- Definir el codi per tal de que el test anterior passi (incloent la creació de la classe PayStationImplRUBLO).

Iteració 8 (10 kopeks RUBLO):

- Definir el test per provar una moneda vàlida de RUBLO (10 kopeks).
- Definir el codi per tal de que el test anterior passi.

Iteració 9 (50 kopeks RUBLO):

- Definir el test per provar una moneda vàlida de RUBLO (50 kopeks).
- Definir el codi per tal de que el test anterior passi.

Iteració 10 (7 kopeks RUBLO):

- Definir el test per provar una moneda invàlida d'EURO (7 kopeks).
- Definir el codi per tal de que el test anterior passi (s'activi una excepció).

Iteració 11 (Integració): A les iteracions anteriors hem definit els test cases per a cada moneda on s'han definit els tests de monedes acceptades i rebutjades. Per una altra banda, a TestPayStation tenim els tests de la resta d'operacions assumint que totes les monedes són vàlides.

- Definir un test case TestIntegration per provar que tots els parquímetres funcionen tal i com s'espera. Definir un test per afegir monedes vàlides (addPayment) per a cada moneda.