

Sessió 1: Introducció a Refactoring

En aquest exercici es pretén donar una idea de les eines més utilitzades per fer refactoring de codi amb IntelliJ IDEA. Es farà a partir del codi base assignat a aquesta sessió en les tasques del [github classroom](#). Per obtenir aquest codi base en un repositori que compartireu les 2 persones de l'equip, cal que accediu a l'adreça:

<https://classroom.github.com/a/yLiOsykV>

Com es el primer cop, cal que us doneu d'alta com a grup en el [github classroom](#) amb el codi que us donarà el/la professor/a a classe.

1. Primers passos:

Per a navegar pel projecte és útil i ràpid utilitzar les tecles enlloc del ratolí. Aquí us deixem les més comuns tot i que IntelliJ té l'ajut de CTRL+Shift+A per a cercar l'acció que es vol fer, sense haver de recordar els menús i també dóna les tecles equivalents a les accions. Aquestes combinacions són per Linux. Trobareu les equivalents a windows i a Mac en aquest [enllaç](#).

Per a inspeccionar el projecte es pot prémer

Alt + 1 per anar al menú del projecte, amb les tecles de moviment per navegar pels fitxers,
<INTRO> per obrir el fitxer i
ESC per anar al fitxer.

Per executar els tests, directament es pot prémer Ctrl+Shift+10.

A continuació s'expliquen els passos a seguir per obrir el projecte bàsic des d'IntelliJ.

1.1. Exploració del projecte videoStore

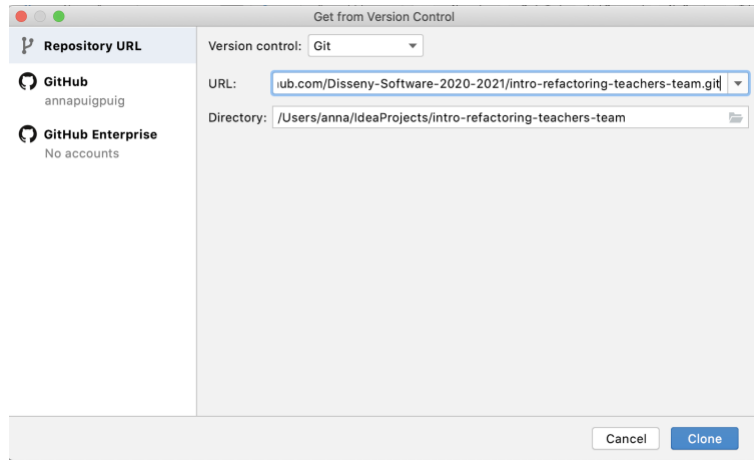
Existeixen dues opcions per obrir el vostre projecte videoStore del github: (a) sincronitzar el vostre projecte local des de dins del IntelliJ amb el github ([Clone Files](#)) (opció recomanada), o (b) baixar el zip i obrir-lo ([Download Files](#)).

- a. [Clone Files](#): L'opció recomanada és File->New->Project From Version Control->Git ... i entrar l'adreça del vostre repositori de la tasca de github classroom.

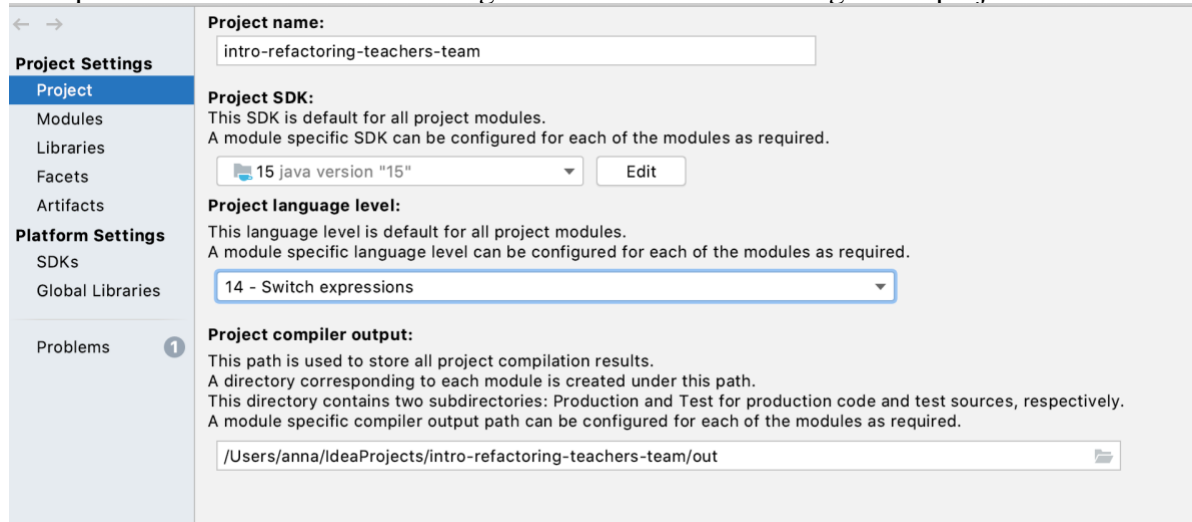
Disseny de Software

Curs 2021-22

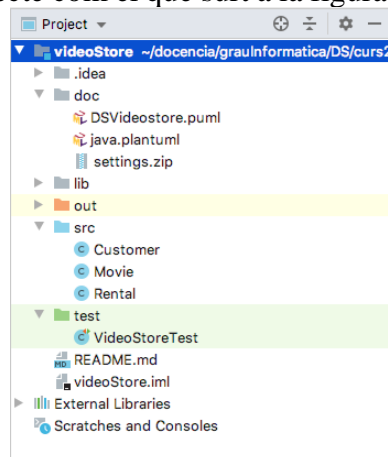
Grau d'Enginyeria Informàtica. Facultat de Matemàtiques i Informàtica. UB



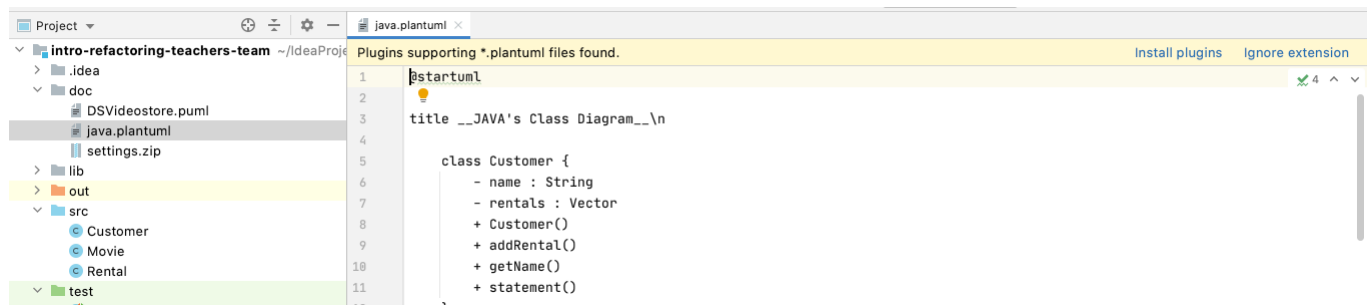
En aquest cas caldrà autenticar-se al github des d'IntelliJ i configurar el projecte amb el SDK:



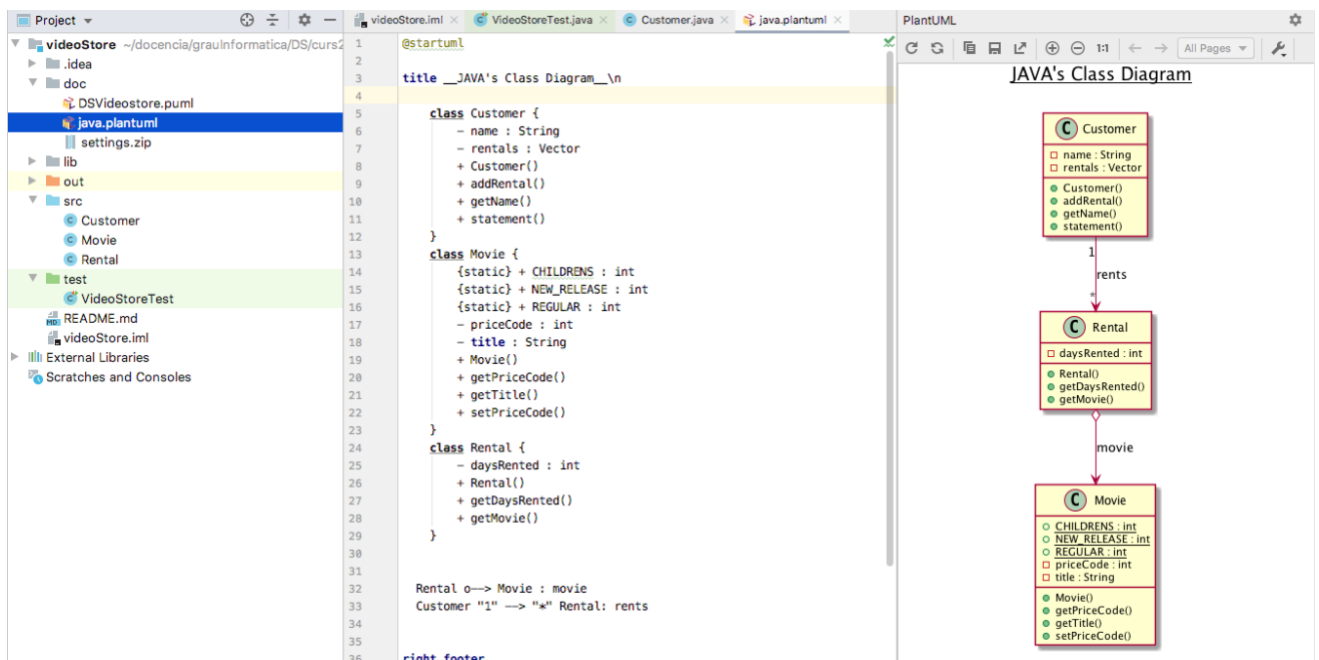
Haurieu de veure un projecte com el que surt a la figura:



- b. **Download Files:** Baixar el projecte videoStore del github de l'adreça <https://classroom.github.com/g/ovqteyCv> i obrir el projecte amb IntelliJ com a projecte a partir de fitxers existents (File->New->Project From Existing Sources...) i afegir la llibreria junit.jar en el pas que us ho demani. Al final marcar els directoris src i test com a Sources Root i Test Root, respectivament (si no ho ha fet automàticament). Us ha de quedar el projecte tal i com es veu a la figura **anterior**.
- c. Instal·lar el plugin **plantUML**: Seleccionar el fitxer java.plantuml de la carpeta doc. Donar al link per a instal·lar el plugin de plantUML 2.0.



Reiniciar IntelliJ. Hauries de veure una visualització del diagrama de classes del projecte en tornar a obrir el fitxer.



Potser en aquest punt no puguem veure bé el diagrama i et surti un missatge com:

```
Dot Executable: /opt/local/bin/dot
File does not exist
Cannot find Graphviz. You should try

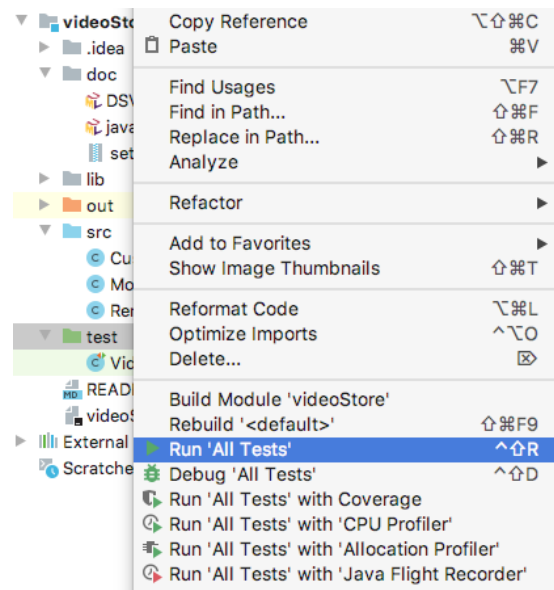
@startuml
testdot
@enduml

or

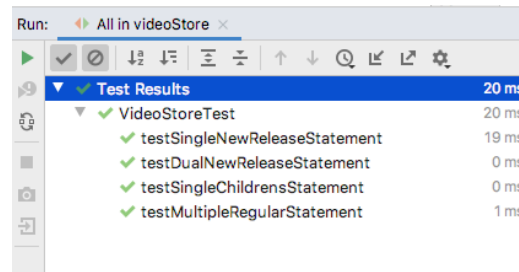
java -jar plantuml.jar -testdot
```

Consulta la pàgina <https://graphviz.org/download/> per a instal·lar-te el graphViz en el teu sistema operatiu. Si tens problemes baixant aquest paquet, en disposes d'un fitxer al campus virtual.

- d. Explora el codi i les classes, mirant el diagrama de classes i el codi associat a cada classe. Per executar els tests. Situar-se a la carpeta de tests i amb el botó dret del ratolí triar Run All Tests... (o directament pots prémer Ctrl+Shift+10)

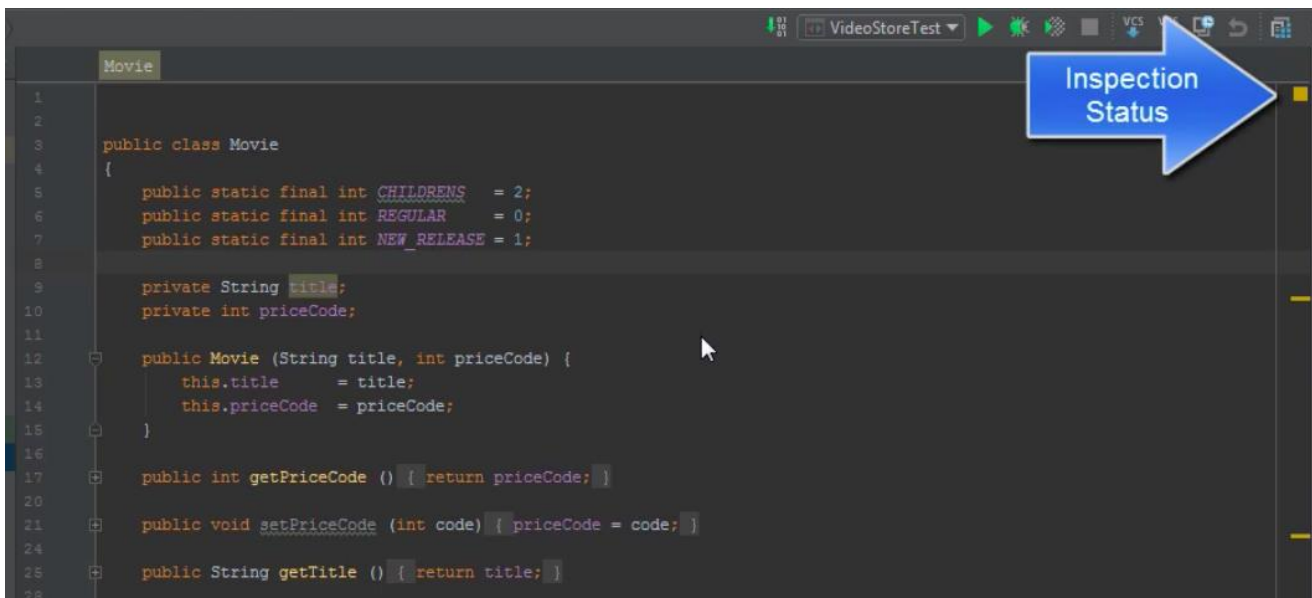


Fixa't que et surten els resultats de l'execució a una finestra de sota, indicant en verd que els tests han passat:

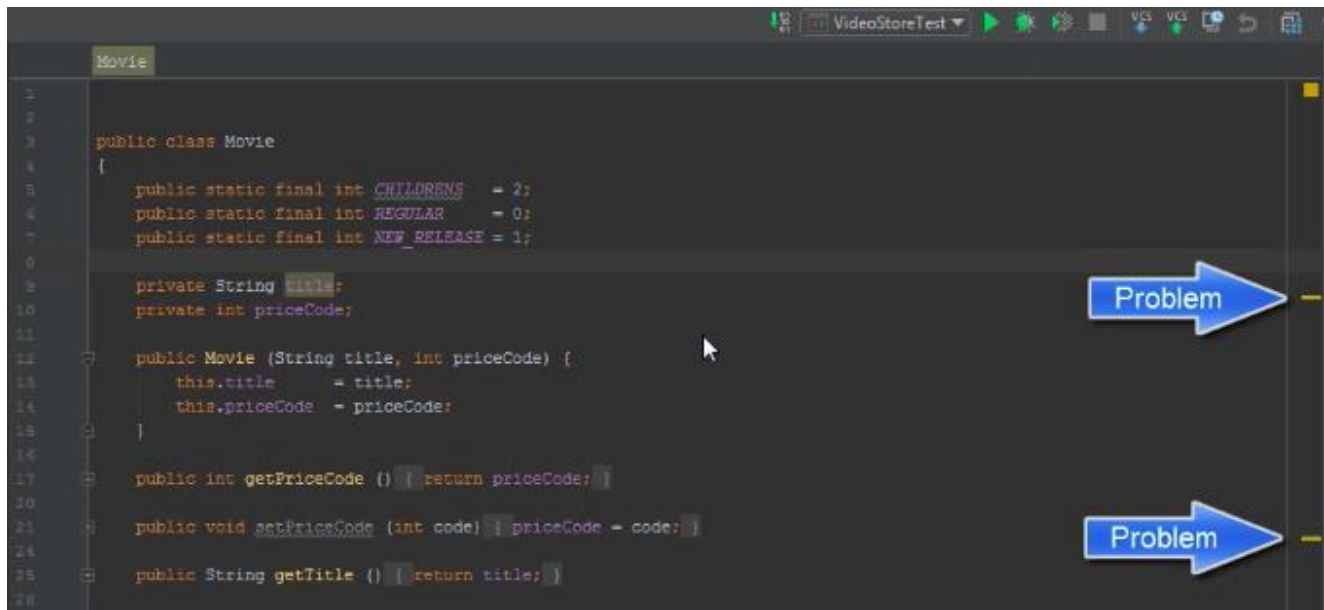


NOTA: Abans de refactoritzar assegura't sempre que tens un bon conjunt de tests que cobreixin els tots els casos i que siguin autocontinguts.

- e. Explorar problemes en les classes: Començar per la classe Movie, anant al punt d'exploració (Inspection Status):

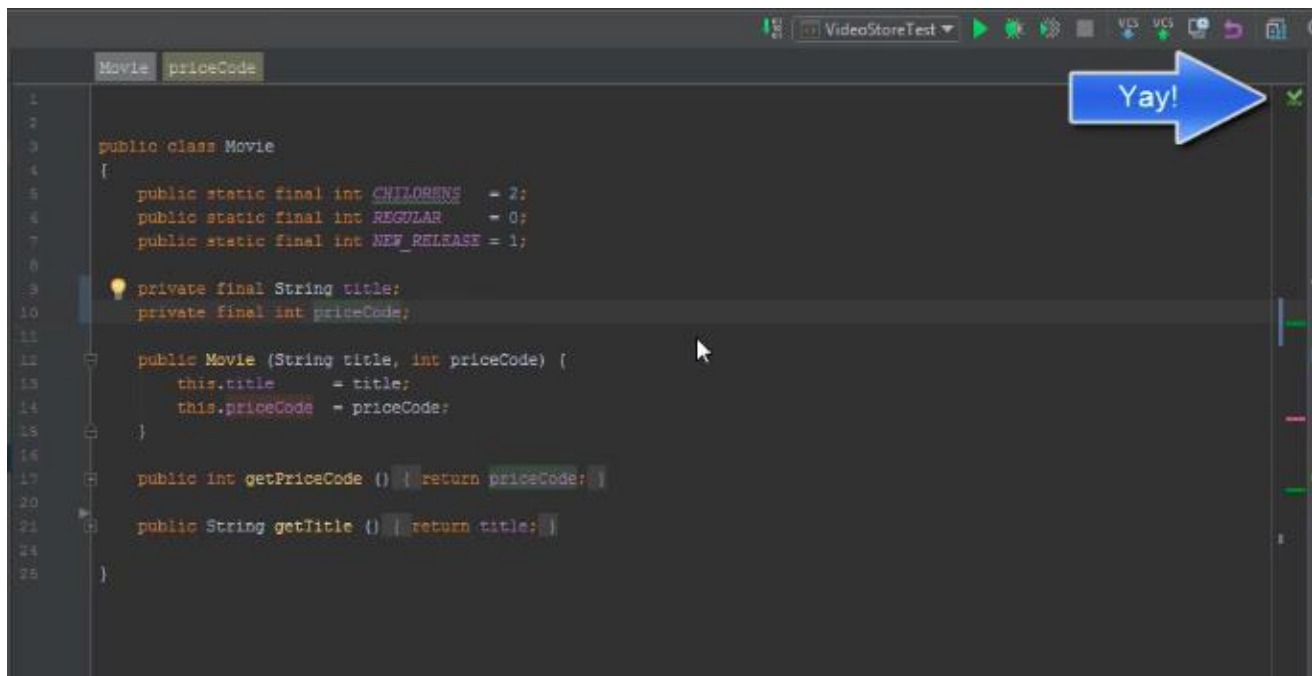


Utilitzar F2 per anar al primer problema i Alt+<INTRO> per a obtenir suggeriments, que no es tenen per què acceptar, però que ara els analitzarem i acceptarem. Per anar al següent problema, tornar a prémer F2.



```
1 public class Movie
2 {
3     public static final int CHILDRENS = 2;
4     public static final int REGULAR = 0;
5     public static final int NEW_RELEASE = 1;
6
7     private String title;
8     private int priceCode;
9
10    public Movie (String title, int priceCode) {
11        this.title = title;
12        this.priceCode = priceCode;
13    }
14
15    public int getPriceCode () { return priceCode; }
16
17    public void setPriceCode (int code) { priceCode = code; }
18
19    public String getTitle () { return title; }
```

Quan tots estan solucionats, surt una marca verda en el punt d'exploració:



```
1 public class Movie
2 {
3     public static final int CHILDRENS = 2;
4     public static final int REGULAR = 0;
5     public static final int NEW_RELEASE = 1;
6
7     private final String title;
8     private final int priceCode;
9
10    public Movie (String title, int priceCode) {
11        this.title = title;
12        this.priceCode = priceCode;
13    }
14
15    public int getPriceCode () { return priceCode; }
16
17    public String getTitle () { return title; }
```

Executar els tests després dels canvis per a validar que segueixen funcionant tots!!

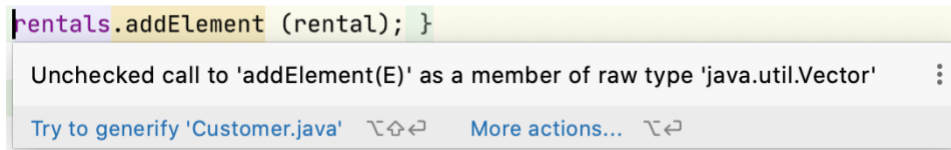
Analitzar la resta de classes per a veure quins suggeriments dona i quins podem acceptar.

Pista: A la classe Customer es pot canviar el Vector generic de lloguers per una declaració més concreta

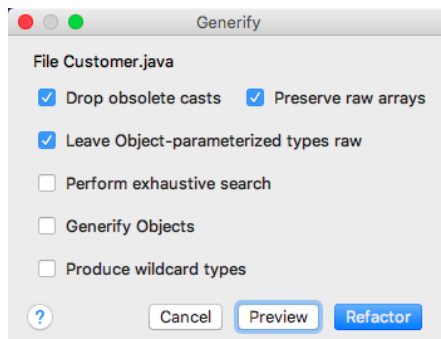
que no calgui fer cast a cada accés, com per exemple:

```
private final Vector<Rental> rentals = new Vector<>();
```

Com es fa? En el suggeriment que indica en fer F2:



Accepteu la generificació de la variable Rental:



Anem ara a la resta de suggeriments. Aneu seguint els consells de la refactorització. Exploreu-los i decidiu si canviar o no el codi.

En la part final es vol acabar canviar la declaració de rentals per:

```
private final List<Rental> rentals = new ArrayList<>();
```

Com ho faries? Fixeu-vos com IntelliJ us suggereix possibilitats d'ajuda: (1) quan poseu només la L de List quan us situeu a la paraula Vector en la declaració de la variable, o (2) en la declaració del new Vector, si us poseu a Vector i premeu CTRL+Espai, us suggereix possibilitats de llistes en Java.

Quan fem aquest canvi, produïm un error a la línia 14 del codi. Mireu amb CTRL+<espai> el suggeriment (fixeu-vos que aquest cop és un error de compilació i no un warning de codi). Fixeu-vos que l'ArrayList no té el mètode addElement i l'he de canviar per add (ha de quedar rentals.add(..)). En fer aquest canvi, a la línia 24 ara dóna error en voler fer una enumeració de l'ArrayList. Cal canviar l'enumeració per un iterador. Després, canvieu l'Enumerator per un Iterador. Moveu la declaració de l'iterador just davant del while. Us donarà algun error de mètode per usar un Iterador i no una enumeració. Modifiqueu el codi per a que no us donin més errors.

```
Iterator<Rental> rentals = this.rentals.iterator();  
while (rentals.hasNext()) {  
    double    thisAmount = 0;  
    Rental    each = rentals.next ();
```

Finalment, si us situeu en el while, us demanarà si el voleu millorar per un for each en primer Alt+<INTRO> (enhanced for).

Proveu a passar de l'enumeració a un codi com

```
for (Rental rental : rentals) {  
    ...  
}
```

Què has d'esborrar? Quines variables tens de més?

Finalment reformatja el fixer prement Ctrl+Alt+Shift+L. Si no ho recordessis sempre pots fer Shit+Crtl+A i buscar Reformat File...

2. Refactoring

2.1. Noves utilitats: escriure la sortida en HTML

Problema: Es vol afegir una nova utilitat que en lloc de donar els missatges per pantalla, els escrigui en HTML en el format següent:

```
<H1> Rentals for <EM> MovieName </EM> </H1><P>  
MovieTitle1: MoviePrice 1<BR>  
MovieTitle2: MoviePrice2 <BR>  
MovieTitle3: MoviePrice3 <BR>  
....  
<P> You owe <EM> totalDebt </EM><P>  
<P> On this rental you earned <EM> totalPoints </EM> frequent renter points<P>
```

Per a afegir aquesta funcionalitat, revisa el mètode statement de la classe Customer. Quins canvis faries?

NOTA:

Quan es va a afegir una nova funcionalitat el codi i es veu que el codi no està ben estructurat per a afegir la nova funcionalitat, primer refactoritza i després afegeix la funcionalitat

2.1.1. Redistribuir el mètode statement de la classe Customer

Comprova que els test passen!

a. Extreure el mètode del càlcul del deute

Primer fes una anàlisi dels paràmetres d'entrada i sortida: les variables que no canvien, les puc passar com a paràmetre d'entrada del nou mètode i si només tinc una que canvia, la puc agafar com a retorn del nou mètode (rental no està modificada en el codi i en canvi thisAmount si).

Selecciona des de la declaració de thisAmount (línia 24) al final del switch (codi que tindrà el nou nom i com a paràmetre rental.

Prem CTRL+Alt+M (o vés al Menu Refactor->Extract->Method) i anomena al mètode nou getRentalPrice.

b. Canvi de nom de la variable thisAmount per price (en el mètode statement), fent Shift+F6 o anant al menu Refactor->Rename

c. Ara aquest nou mètode ja no té sentit en aquesta classe, ja que no fa res amb els atributs de Customer. Canviarem el mètode a la classe Rental.

Per a fer un canvi de mètode a una altra classe, sempre cal tenir un paràmetre del mètode del tipus de la classe on volem moure el mètode per a que IntelliJ ens pugui ajudar en aquest procés. En aquest cas ja el tenim (paràmetre rental de tipus Rental), però en el cas que no el tinguéssim primer hauríem d'afegir un paràmetre de tipus la classe on es vol moure (usant Refactor->Change Signature).

Ara premem F6 (Refact->MoveInstanceMethod) i seleccionem la classe Rental com a destí. Podem veure previament el que es farà en el refactoring abans d'aplicar-lo. L'apliquem i veurem que s'obre la classe Rental, mostrant on s'ha mogut el mètode i com ha canviat.

d. Podem també extreure la variable temporal price (del mètode statement) per la seva crida directa al mètode rental.getRentalPrice(). Això ho podem fer extraient un mètode de la crida rental.getRentalPrice per anomenar-se getAmountFor de la classe Customer amb Refactor->Replace Temp with Query.. quedant el codi com:

```
class Customer....
```

```
private double getAmountFor(Rental rental) {
    return rental.getRentalPrice();
}

public String statement() {
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    String result = "Rental Record for " + getName() + "\n";

    for (Rental rental : rentals) {

        frequentRenterPoints++;

        if (rental.getMovie().getPriceCode() == Movie.NEW_RELEASE
            && rental.getDaysRented() > 1)
            frequentRenterPoints++;

        result += "\t" + rental.getMovie().getTitle() + "\t"
            + String.valueOf(getAmountFor(rental)) + "\n";
        totalAmount += getAmountFor(rental);

    }
    result += "You owed " + String.valueOf(totalAmount) + "\n";
    result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter points\n";

    return result;
}
```

Executa de nou els tests per a comprovar que tot funciona.

e. Treu el càlcul de totalAmount a un nou mètode getTotalCharge (Refactor->Replace Temp by

Query). Fixa't que ara totalAmount es modifica en el loop i Replace Temp with Query no funciona directament. Aquí pots fer:

e.1. Treu la línia del loop totalAmount += getAmountFor(rental) que crea la dependència d'escriptura

e.2. Posa't sobre la variable totalAmount del final del loop i fes el Replace Temp with a Query... amb un nou mètode anomenat getTotalCharge i copia el loop amb el càlcul que has tret de totalAmount i afegeix la instrucció “return totalAmount” després del loop. Accepta els suggeriments de crear una variable local, d'inicialitzar-la, etc.. fins a trobar el mètode següent.

```
private double getTotalCharge() {  
    double totalAmount = 0;  
    for (Rental rental : rentals) {  
        totalAmount += getAmountFor(rental);  
    }  
    return totalAmount;  
}
```

e.2. Comprova que els tests passen.

f. Fes els passos similars per a la part del mètode statement que calcula els Frequent Renter Points

- Extract Method a un mètode anomenat calculatePoints que sigui tota la part del if

- Esborra el paràmetre int d'entrada del mètode calculatePoints fent Refactor->Change signature. Això introduirà un error al mètode per què la variable frequentRenterPoints no està inicialitzada, prem F2 i vés a l'error i amb Alt+<INTRO> accepta el suggeriment de declarar-la i inicialitzar-la. Ara compila i executa els tests..... No funciona? Revisa l'assignació de la crida a calculatePoints... Ha de ser:

```
frequentRenterPoints += calculatePoints(rental);
```

Comprova que passa ara els tests...

- Mou el mètode calculatePoints a la classe Rental

- Treu el càlcul del total de frequentRenterPoints anomenat getTotalFrequentRenterPoints de la classe Customer, de forma similar al que has fet amb el mètode getTotalCharge

```
private int getTotalFrequentRenterPoints() {  
    int frequentRenterPoints = 0;  
    for (Rental rental : rentals) {  
        frequentRenterPoints += rental.calculatePoints();  
    }  
}
```

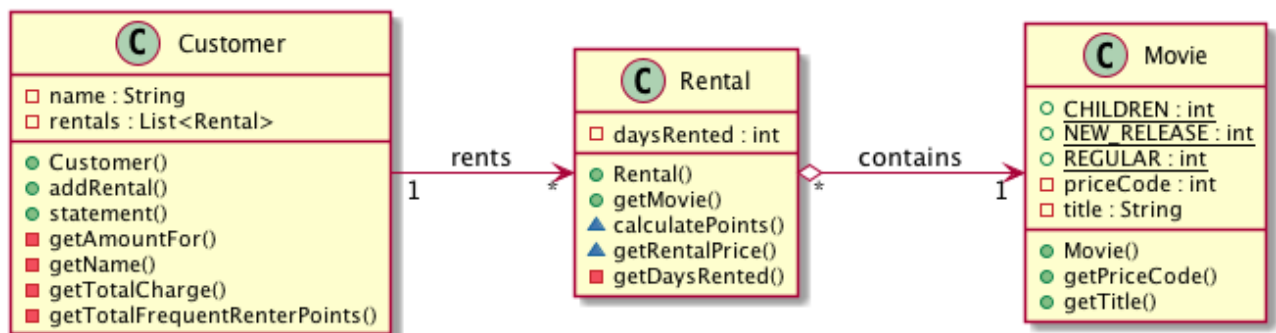
```

    return frequentRenterPoints;
}

```

Ara, quan passin els tests, ja estàs a punt per a introduir la nova funcionalitat. Hauries de tenir el següent diagrama de classes:

SRC's Class Diagram abans d'incloure htmlStatement



PlantUML diagram generated by SketchIt! (<https://bitbucket.org/pmesmeur/sketch.it>)
For more information about this tool, please contact philippe.mesmeur@gmail.com

Fes CTRL+Alt+Shift+L per a reordenar el codi.

2.1.2. Introduir el nou mètode htmlStatement a la classe Customer

Comprova que els tests passen. Ara fés el nou mètode htmlStatement que imprimeix diferent el codi. Per això cal que facis primer els tests i després el mètode en la classe Statement. Segeueix la mateixa estructura que tens ara del codi.

a. Fes el nou test de la classe VideoStoreTest:

```

public void testSingleNewReleaseHtmlStatement () {

    customer.addRental (new Rental(new Movie("The Cell", Movie.NEW_RELEASE), 3));

    assertEquals("<H1>Rentals for <EM>Fred</EM></ H1><P>\nThe Cell: 9.0<BR>\n<P>You
owe <EM>9.0</EM><P>\nOn this rental you earned <EM>2</EM> frequent renter

```

```
points<P>", customer.htmlStatement ());  
  
}
```

- b. Ara implementa el mètode `htmlStatement` a la classe `Customer`.
- c. Executa els tests per a verificar que funciona correctament

2.2. Noves utilitats: nous tipus de pel·lícula

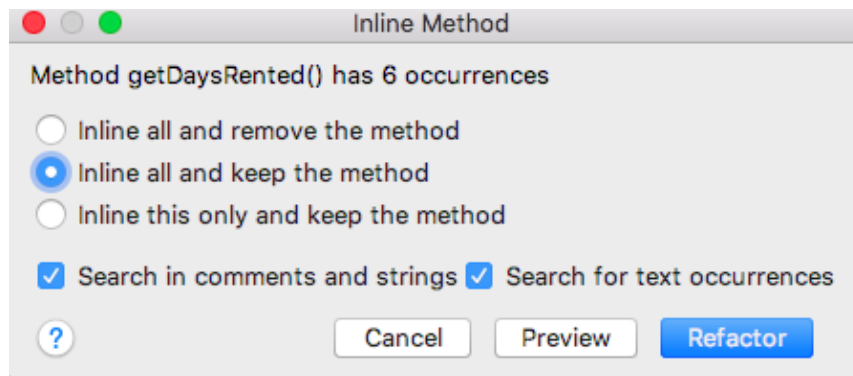
Ara el client comenta que voldrà en un futur nous tipus de pel·lícules encara que no sap ben bé els tipus que hi hauran. Sabent que això és un canvi possible a tenir en compte en el codi, anem a mirar de canviar-lo per a suportar-lo en un futur.

2.2.1. Canviar el switch per ús de polimorfisme

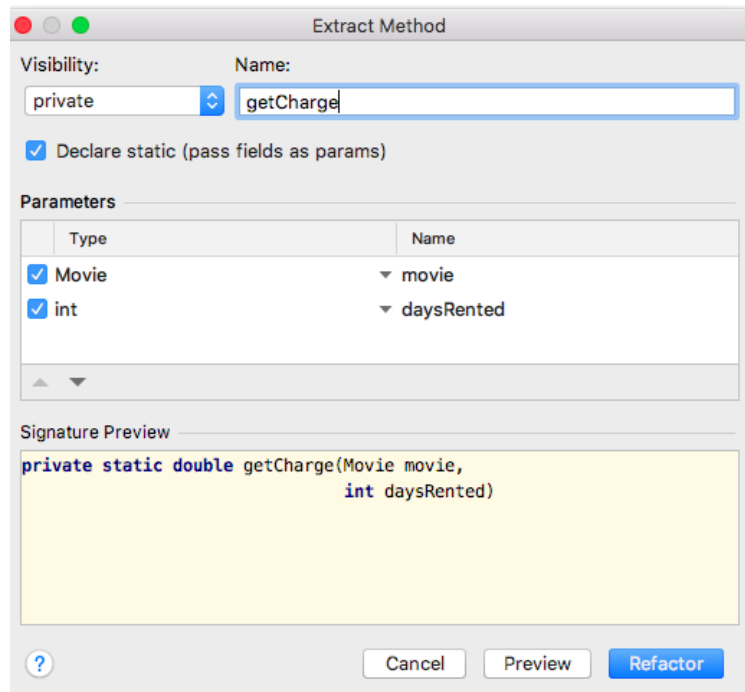
El que ens està limitant el tipus de pel·lícula o el codi a canviar quan entri un nou tipus de pel·lícula serà el switch del mètode `getRentalPrice`.

El que volem primer es construir un mètode anomenat `getCharge(int daysRented)` que contingui el switch i que sigui de la classe `Movie`. Per fer això podem seguir els següents passos:

- a. Fer que totes les crides al mètode `getDaysRented` siguin directament l'atribut de la classe. Això es fa col·locant el cursor en qualsevol de les crides i fent `Refactor->InlineMethod` i substituint totes les crides però conservant el mètode en la classe:



- b. Fer el mateix amb el `getMovie` (no facis cas de l'avís que et dona IntelliJ quan vols fer aquesta operació).
- c. Extreu el Mètode `getCharge` seleccionant tot el codi del mètode actual `getRentalPrice` i marca que vols passar els atributs com a paràmetre:



d. Mou el nou mètode `getCharge` a la classe `Movie` situant-te en la el nom del mètode i fent `Convert to Instance Method...`

e. Ara, al constructor de la classe `Movie`, on es posa el preu en l'atribut `this.priceCode` inclou un nou mètode (`setPriceCode`), seleccionant tota la línia `"this.priceCode = priceCode;` i cridant a `Extract Method` per a fer un `setPriceCode` en el constructor de la classe `Movie`

```
private final String title;
```

```
private int priceCode;
```

```
public Movie (String title, int priceCode) {
```

```
    this.title    = title;
```

```
    setPriceCode(priceCode);
```

```
}
```

f. Prova a fer els gets i els sets de l'atribut `priceCode`, situant-te en l'atribut i fent `Refactor->EncapsulateFields...`

g. Per a fer el polimorfisme hem de crear la classe abstracte `Price` i els seus fills amb el mètodes de cadascuna de les classes que donaran els preus de cada tipus

Crea la classe Price fent servir l'ajuda de IntelliJ, posant un atribut a la classe Movie de tipus Price i després acceptant el suggeriment de crear una classe. Afegeix-li el mètode abstracte getPriceCode() i implementa'l a cadascuna de les classes filles (Children, NewRelease i Regular).

```
abstract class Price {  
    abstract int getPriceCode();  
}  
  
class ChildrenPrice extends Price {  
    @Override  
    int getPriceCode() {  
        return Movie.CHILDREN;  
    }  
}
```

Ara esborrarem l'atribut priceCode de la classe Movie i deixarem els seus get i set per a que facin servir el nou atribut price.

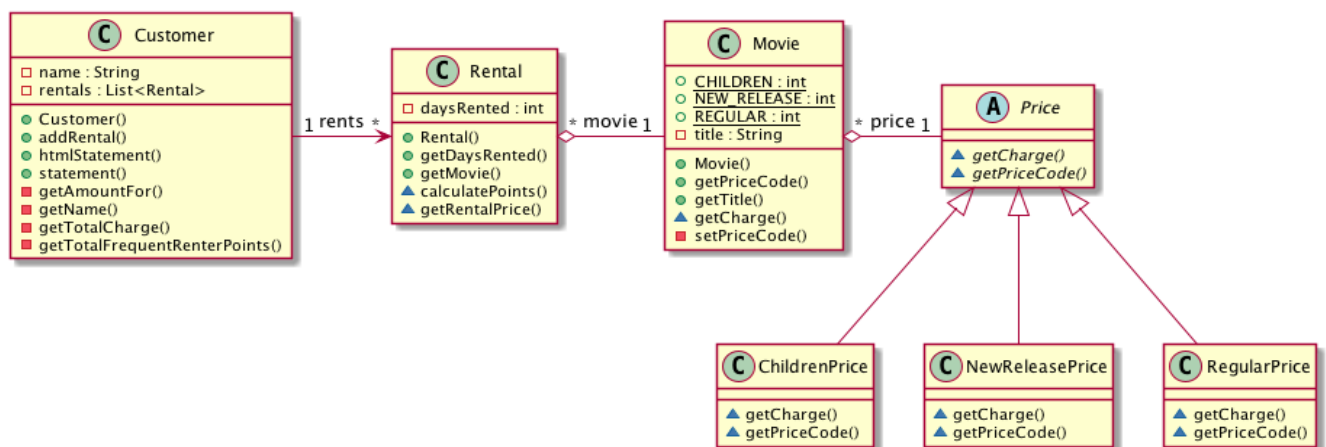
```
private void setPriceCode(int priceCode) {  
    switch (priceCode) {  
        case Movie.NEW_RELEASE:  
            price = new NewReleasePrice();  
            break;  
        case Movie.REGULAR:  
            price = new RegularPrice();  
            break;  
        case Movie.CHILDREN:  
            price = new ChildrenPrice();  
            break;  
        default:  
            throw new IllegalStateException("Unexpected value: " + priceCode);  
    }  
}
```

```
public int getPriceCode () {
    return price.getPriceCode();
}
```

Fes un nou mètode abstracte a la classe Price que sigui `getCharge(int rentedDays)` i estigui implementat a les classes filles, traient el switch de la classe Movie. Usant les utilitats que has après i còpia el codi en el cas de moure el codi concret a cadascuna de les classes filles.

Al final el teu diagrama de classes hauria de quedar com el següent:

SRC's Class Diagram

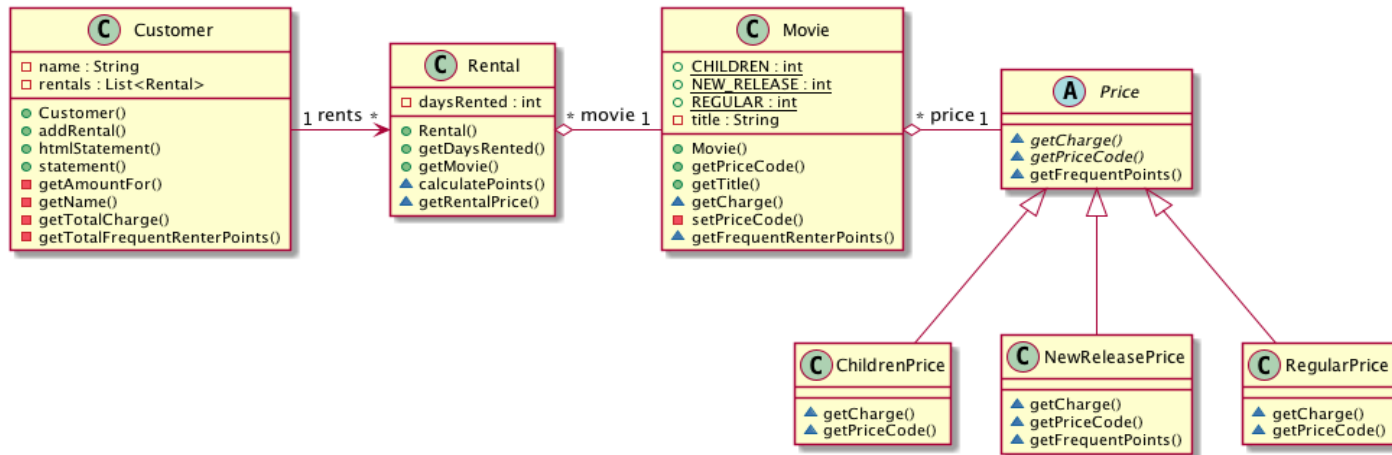


PlantUML diagram generated by SketchIt! (<https://bitbucket.org/pmesmeur/sketch.it>)
For more information about this tool, please contact philippe.mesmeur@gmail.com

Fes el mateix exercici per a `getFrequentRenterPoints`. Fixa't que en aquest cas només canvia el mètode en les pel·lícules `NewRelease` i no en les altres? Serà un mètode abstracte de la classe `Preu`?

Aquesta hauria de ser la teva solució final:

SRC's Class Diagram



PlantUML diagram generated by SketchUML (<https://bitbucket.org/pmameur/sketchuml>)
For more information about this tool, please contact philippe.mesmeur@gmail.com

El que acabes de fer es substituir un condicional per polimorfisme. Això es pot fer manualment com ho has anat fent, usant les eines bàsiques de IntelliJ però també es poden usar altres estratègies dins d'IntelliJ com la descrita a l'enllaç següent, si tens ganes de saber-ne més <https://www.jetbrains.com/help/idea/replace-conditional-logic-with-strategy-pattern.html>