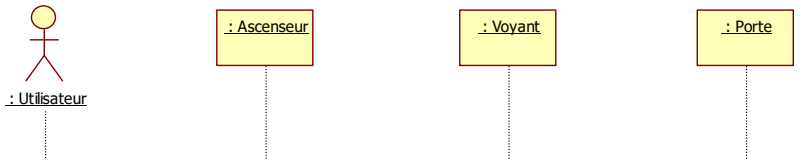
M2104 - TD n°3 : Diagramme de séquences

Exercice 1 : Un petit échauffement sur le diagramme de séquence…

Un dispositif d'**ascenseur** dispose d'un **voyant** qui est ***éteint*** lorsque l'ascenseur est disponible et ***allumé*** lorsque l'ascenseur est en train de se déplacer. L'ascenseur dispose également d'une **porte** qui permet à un utilisateur d'entrer et de sortir de l'ascenseur : l'ascenseur ne peut se déplacer que lorsque la porte est ***fermée***.
L'ascenseur doit également connaître sa ***position***, c.-à-d. l'étage auquel il se trouve.

1.a Proposez un diagramme de classes afin de modéliser la description précédente.

1.b. Lorsque l'on souhaite utiliser un ascenseur dans 80% des cas, l'ascenseur est disponible et sa porte est fermée. Le scénario suivant va alors se produire. L'utilisateur va appeler l'ascenseur. L'ascenseur va allumer son voyant, puis il va ensuite se déplacer jusqu'à l'étage de l'utilisateur. Arrivé au bon étage, l'ascenseur va ouvrir sa porte afin de laisser entrer l'utilisateur. Une fois dans l'ascenseur, l'utilisateur peut choisir un étage. L'ascenseur va alors fermer la porte, puis se déplacer jusqu'à l'étage souhaité. Une fois arrivé, l'ascenseur ouvrira la porte et éteindra le voyant.
A partir de la description précédente, vous allez devoir représenter le diagramme de séquence correspondant à l'utilisation nominale d'un ascenseur.
Le squelette du diagramme de séquence attendu vous est donné ci-dessous : il ne vous reste plus qu'à rajouter les messages sur ce diagramme.



1.c. ***Le diagramme de séquences à la conquête des opérations du diagramme de classes…***
A partir du diagramme de séquences établi dans la question précédente, enrichissez le diagramme de classes construit à la question 1.a

1.d Proposez un ***diagramme de communication*** correspondant au diagramme de séquences obtenu précédemment. Quel est l'intérêt du diagramme de communication en regard du diagramme de classes ?

1.e ***Utilisation de fragment combiné (UML 2.0)***
Transformez votre diagramme de séquence afin de faire apparaître qu'après l'appel de l'utilisateur et l'allumage du voyant, l'ascenseur se déplacera seulement s'il ne se trouve pas au même étage que l'utilisateur...

Exercice 2 : Un peu de reverse-engineering …

… de l'implémentation à la conception : diagrammes de classes et diagramme de séquence comme outils de reverse engineering rescousse pour une autre vision sur le code…

(Le code ci-dessous est extrait de l'exemple du premier chapitre du livre de Martin Fowler « Refactoring Improving the Design of Existing Code ». Toutefois, il s'agit d'une version simplifiée du code initial puisqu'il gère uniquement le montant des frais de location pour simplifier le TD, et ne prend pas en compte la gestion des points de fidélité).

Le projet **VideoStore** permet de calculer et d'afficher le relevé de compte d'un client d'un magasin de location de vidéos.

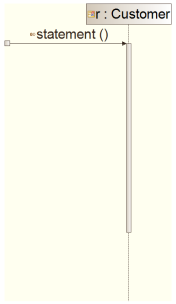
```
-------------------------------------------------------------------------------
public class Movie {
   public static final int REGULAR = 0;
   public static final int NEW_RELEASE = 1;

   private String title;
   private int priceCode;

   public Movie(String title, int priceCode) {
          this.title = title;
          this.priceCode = priceCode;
   }

   public int getPriceCode() {
          return priceCode;
   }

   public void setPriceCode(int i) {
          priceCode = i;
   }

   public String getTitle() {
          return title;
   }
}

-------------------------------------------------------------------------------


public class Rental {
   private Movie movie;
   private int daysRented;

   public Rental(Movie movie, int daysRented) {
          this.movie = movie;
          this.daysRented = daysRented;
   }

   public int getDaysRented() {
          return daysRented;
   }

   public Movie getMovie() {
          return movie;
   }
}

-------------------------------------------------------------------------------
```

```
public class Customer {
   private String name;
   private List<Rental> rentals = new ArrayList<Rental>();

   public Customer(String name) {
         this.name = name;
   }

   public String getName() {
         return name;
   }

   public void addRental(Rental rental) {
         this.rentals.add(rental);
   }

   public String statement() {

         double totalAmount = 0;

         String result = "Rental Record for " + this.getName() + "\n";

         for (Rental each : rentals) {

               double thisAmount = 0;

               // determine amounts for each line
               switch (each.getMovie().getPriceCode()) {
               case Movie.REGULAR:
                     thisAmount += 2;
                     if (each.getDaysRented() > 2) {
                           thisAmount += (each.getDaysRented() - 2) * 1.5;
                     }
                     break;
               case Movie.NEW_RELEASE:
                     thisAmount += each.getDaysRented() * 3;
                     break;
               }
               // show figures for this rental
               result += "\t" + each.getMovie().getTitle() + "\t" + thisAmount + "\n";
               totalAmount += thisAmount;
         }
         // add footer lines
         result += "Amount owed is " + totalAmount + "\n";
         return result;
   }
}
```

---

1. Modéliser le **diagramme de classes** correspondant au code ci-dessus.

2. Représenter sur un **diagramme de séquences** les interactions (messages échangés entre les objets) lors de l'exécution de méthode **`statement`**. Ce diagramme commencera de la manière suivante.
   A vous de le compléter avec les objets et messages appropriés.
   Demandez-vous aussi que fait ce programme et quelles en sont les règles métiers ?



3. 3.a Le diagramme de séquence soulève-t-il des **problèmes de conception**. Lesquels ?

   3.b Nous venons d'apprendre que le client avait une nouvelle exigence : il souhaite absolument que **le relevé de compte puisse également être disponible au format HTML.** Expliquez comment vous procéderiez pour mettre en place cette nouvelle fonctionnalité ? Et quel impact l'ajout de cette nouvelle fonctionnalité pourrait avoir sur votre code ?

4. … Le client vient juste de nous recontacter pour nous indiquer qu'il souhaitait également pouvoir disposer d'un **système de tarification évolutif et personnalisable** c-a-d que les deux tarifications ***REGULAR*** et ***NEW_RELEASE*** ne lui suffisent plus et qu'elles sont trop restrictives à son goût pour ses affaires. Il souhaite donc disposer que le code soit facilement modifiable pour pouvoir étendre rapidement et simplement le système de tarifications à un nombre (indéterminé pour le moment) de tarifs ayant chacun sa propre règle de calcul…

Heureusement, les développeurs du projet **VideoStore** sont des développeurs professionnels… Ils ont bien compris que la conception actuelle du programme n'est pas idéale pour répondre aux nouvelles attentes du client (ajout d'un affichage HTML et d'un système de tarification évolutif).

- ➔ Dans un premier temps, ils ont donc mis en place, une phase de refactoring pour isoler le calcul du montant des frais d'une location de l'affichage du relevé.
  Un refactoring (remaniement) consiste à changer la structure interne d'un logiciel sans en changer son comportement observable (M. Fowler)

- ➔ Ils ont ensuite ajouté la méthode **htmlstatement** pour éditer un relevé au format HTML.

- ➔ Et enfin, ils ont mis en place, le système de tarification flexible.

Après toutes ces modifications (réalisées bien sûr en toute confiance grâce à la présence de tests unitaires automatisés dont le code se trouve en annexe, puisque pour cette question, seul le diagramme autour des classes métier nous intéresse), les développeurs de **VideoStore** propose le code de production suivant *(ce sont bien sûr des développeurs professionnels avec quelques années d'expérience, on n'attend pas de vous d'arriver aujourd'hui à proposer seul une telle conception : un module de conception avancée vous attend l'année prochaine et vous aidera à progresser dans ce sens…)*

```
public class Movie {
      public static final int REGULAR = 0;
      public static final int NEW_RELEASE = 1;

      private String title;
      private Price price;

      public Movie(String title, int priceCode) {
            this.title = title;
            setPriceCode(priceCode);
      }

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

      public void setPriceCode(int priceCode) {
          switch (priceCode) {
          case REGULAR:
              price = new RegularPrice();
              break;
          case NEW_RELEASE:
              price = new NewReleasePrice();
              break;
          default:
              throw new IllegalArgumentException("Incorrect Price Code");
          }
      }

      public String getTitle() {
            return title;
      }

      public double getCharge(int numberOfDaysRented) {
            return price.getCharge(numberOfDaysRented);
      }
}
```

```
public class Rental {
```

```
	private Movie movie;
	private int daysRented;

	public Rental(Movie movie, int daysRented) {
		this.movie = movie;
		this.daysRented = daysRented;
	}

	public int getDaysRented() {
		return daysRented;
	}

	public Movie getMovie() {
		return movie;
	}

	public double getCharge() {
		return movie.getCharge(daysRented);
	}
}
```

```
public class Customer {
	private String name;
	private List<Rental> rentals = new ArrayList<Rental>();

	public Customer(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public void addRental(Rental rental) {
		this.rentals.add(rental);
	}

	public String statement() {

		String result = "Rental Record for " + this.getName() + "\n";

		for (Rental each : rentals) {
			result += "\t" + each.getMovie().getTitle() + "\t"
					+ each.getCharge() + "\n";
		}

		// add footer lines
		result += "Amount owed is " + getTotalCharge() + "\n";
		return result;
	}

	public String htmlStatement() {

	    String result = "<h1>Rental Record for <em>" + this.getName()
	    		+ "</em></h1><p>\n";

	    result +="<table><tboby>";
	    for (Rental each : rentals) {
	        result +=  "<tr><td>"+each.getMovie().getTitle() + "</td><td>"
	                + each.getCharge() + "</tr></td>";
	    }
	    result +="</table></tboby>";

	    result += "<p>Amount owed is <em>" + getTotalCharge() + "</em><p>\n";
	    return result;
	}


	private double getTotalCharge() {
```



```
		double totalCharge = 0;
		for (Rental each : rentals) {
			totalCharge += each.getCharge();
		}
		return totalCharge;
	}
}
```

```
public abstract class Price {
    public abstract int getPriceCode();
    public abstract double getCharge(int numberOfDaysRented);
}
```

```
public class RegularPrice extends Price {
    @Override
    public int getPriceCode() {
        return Movie.REGULAR;
    }

    @Override
    public double getCharge(int numberOfDaysRented) {
        double result = 2;
        if (numberOfDaysRented > 2) {
            result += (numberOfDaysRented - 2) * 1.5;
        }
        return result;
    }
}
```

```
public class NewReleasePrice extends Price {
    @Override
    public int getPriceCode() {
        return Movie.NEW_RELEASE;
    }

    @Override
    public double getCharge(int numberOfDaysRented) {
        return numberOfDaysRented * 3;
    }
}
```

4.a Modéliser le **diagramme de classes** de ce projet ?

4.b Représenter sur un **diagramme de séquences** les interactions (messages échangés entre les objets) lors de l'exécution de méthode **`statement`**.

4.c A partir de ces diagrammes, répondez aux questions suivantes :

- Le refactoring a permis de séparer le traitement lié à l'affichage du relevé, du traitement lié au calcul du montant des frais de location. Quelle classe **est responsable de l'affichage** du relevé ?

- Quelle classe est **responsable du calcul du montant total du relevé** (somme de tous les frais de location) ?

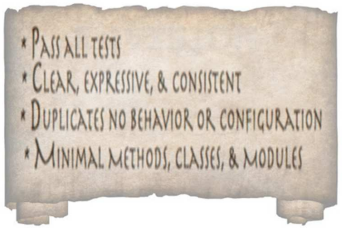
- Quelle classe est **responsable du calcul des frais d'une location** ?

- Quelle classe est **garante (responsable) de la règle de calcul** des frais d'une location ?

- Comment feriez-vous évoluer le système de tarification pour qu'**un nouveau tarif de type** `FABULOUS_STUDENT_PRICE` soit proposé aux étudiants à 1 Euro le premier jour, puis 0.5 euros pour les jours suivants.

Remarques :

➔ A la lecture du code précédent et visualisant la conception du projet à travers les diagrammes de classes et de séquence, vous pouvez remarquer que les développeurs craftsmen de **VideoStore** se sont appliqués à développer leur projet autour d'un code propre et d'une **C**onception **O**rientée **O**bjet "**simple**" (au sens de **Simple Design**) c-a-d qui respecte les 4 règles ci-dessous (image extraite de : http://wiki.c2.com/?XpSimplicityRules) :



Autrement dit :

- Tous les tests passent
- un code qui révèle son intention
- pas de duplication
- des éléments « les plus petits possibles » pour respecter au mieux les trois règles précédentes (diviser pour réduire la complexité…)

➔ Attention, il n'est pas simple de *bien* concevoir un système, il faut un peu (beaucoup ?) d'expérience et ne pas trop s'éloigner du code et de la technique. D'ailleurs, ne dit-on pas qu'un architecte logiciel doit avant tout un très bon développeur ? et, oui un réel débat existe bel et bien sur ce sujet … Mais, c'est une autre histoire…

ANNEXE : Code de test du projet Video Store (pour information)

```
public class CustomerTest {

    @Test
    public void statementForOneRegularMovie() {
        Customer customer = new Customer("Alice");
        Movie movie = new Movie("The Lord of the Rings", Movie.REGULAR);
        Rental rental = new Rental(movie, 3); // 3 days rental
        customer.addRental(rental);

        String statement = customer.statement();

        String expected = "Rental Record for Alice\n" +
                "\tThe Lord of the Rings\t3.5\n" +
                "Amount owed is 3.5\n";
        assertEquals(expected, statement);
    }


    @Test
    public void statementForOneNewReleaseMovie() {
        Customer customer = new Customer("Bob");
        Movie movie = new Movie("Star Wars", Movie.NEW_RELEASE);
        Rental rental = new Rental(movie, 3); // 3 day rental
        customer.addRental(rental);

        String statement = customer.statement();

        String expected = "Rental Record for Bob\n" +
                "\tStar Wars\t9.0\n" +
                "Amount owed is 9.0\n";
        assertEquals(expected, statement);
    }


    @Test
    public void statementForManyMovies() {
        Customer customer = new Customer("Bob");
        Movie movie1 = new Movie("Star Wars", Movie.NEW_RELEASE);
        Rental rental1 = new Rental(movie1, 2); // 2 day rental
        Movie movie2 = new Movie("The Lord of the Rings", Movie.REGULAR);
        Rental rental2 = new Rental(movie2, 1); // 1 day rental
        customer.addRental(rental1);
        customer.addRental(rental2);

        String statement = customer.statement();

        String expected = "Rental Record for Bob\n" +
                "\tStar Wars\t6.0\n" +
                "\tThe Lord of the Rings\t2.0\n" +
                "Amount owed is 8.0\n";
        assertEquals(expected, statement);
    }

    @Test
    public void htmlStatementForOneRegularMovie() {
        Customer customer = new Customer("Alice");
        Movie movie = new Movie("The Lord of the Rings", Movie.REGULAR);
        Rental rental = new Rental(movie, 3); // 3 day rental
        customer.addRental(rental);

        String htmlStatement = customer.htmlStatement();

        String expected = "<h1>Rental Record for <em>Alice</em></h1><p>\n"
                + "<table><tboby><tr><td>The Lord of the
Rings</td><td>3.5</tr></td></table></tboby>"
                + "<p>Amount owed is <em>3.5</em><p>\n";
        assertEquals(expected, htmlStatement);
    }
}
```