



Cursus JAVA

M2I Formations 2022

Alan Piron-Lafleur



MODULE JAVA

JAVA POO

Alan Piron-Lafleur



1.

Les bases du Java



Lancer Eclipse

- Lancer Eclipse
- Fermer la welcome page
- Aller dans windows -> preferences
- Aller dans General -> workspace pour mettre le text encoding en ISO
- Faites “Apply and Close”
- Create a Java project
- Nommez le projet (M2iPOO)
- Décocher “create module.info”
- Cliquez sur “finish”



Le fameux HelloWorld !

```
package module1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World !");
    }

}
```

La notion de package ?

Le fichier doit avoir le même nom que la classe

Exercice :

- Créer un nouveau package (clic droit sur SRC)
- Nommer le module "module1"
- Clic droit sur le package -> new -> Class
- Rentrer un nom de classe (HelloWorld) et cocher la case pour créer un main()
- Afficher un « Hello World »

L'affichage de messages sur la console

- Sortie standard
 - `System.out.println()` : *retour à la ligne*
 - `System.out.print()` : *sans retour à la ligne*
- Sortie des messages d'erreur
 - `System.err.println()`
 - `System.err.print()`

Les variables et constantes

```
public class Taxe {  
  
    public static final double TVA = 20.0/100;  
  
    public static void main(String[] args) {  
        double prixHT = 17.85;  
        System.out.println("Prix TTC de l'article : " + prixHT*(1+TVA));  
    }  
}
```



Prix TTC de l'article : 21,42

Les tableaux

- Déclaration

```
int[] valeurs = {45,12,22,66,98};
```

- Écriture dans une case

```
valeurs[2] = 1792;
```

- Lecture d'une valeur

```
System.out.println(valeurs[2])
```



Les indices des
éléments d'un tableau
commencent à zéro

Exercice :

Réaliser l'exemple.

Vous devez afficher « 1792 » à la fin.

La récupération des saisies de l'utilisateur

- Utilisation de Scanner
 - `nextLine()` : Lecture d'une chaîne de caractères
 - `nextInt()` : Lecture d'un nombre entier
 - `nextFloat()` : Lecture d'un nombre à virgule
 - ...

- Exemple :

```
Scanner s = new Scanner(System.in);
System.out.println("Quel est votre prénom ?");
// saisir la chaîne de caractères entrée
// par l'utilisateur
String prenom = s.nextLine();
s.close();
```

Exercice :

Réaliser l'exemple.

Egalement, je veux que le prénom saisi par l'utilisateur s'affiche à la fin.



Les conditionnelles

- L'instruction if (simple)

```
if(prenom.length()==0)
    System.err.println("Le prénom est obligatoire !");
```

- L'instruction if (double)

```
if(prenom.length()==0)
    System.err.println("Le prénom est obligatoire !");
else
    System.out.println("Bonjour " + prenom);
```

Accolades si plusieurs instructions

```
if(prenom.length()==0) {
    System.err.println("Le prénom est obligatoire !");
    System.exit(1);
}
```

Exercice :

Concevoir un programme qui :

- Demande son prénom à l'utilisateur
- Si aucun prénom n'est tapé : affiche un message d'erreur
- Si un prénom est tapé : « Ok, c'est toi prénom » (remplacer 'prénom' par le prénom préalablement saisie)

Les conditionnelles

- L'instruction switch

```
switch (aJeter) {  
    case "boite de conserve":  
        System.out.println("Recyclage");  
        break;  
    case "épluchures":  
        System.out.println("Composte");  
        break;  
    default:  
        System.out.println("Poubelle");  
        break;  
}
```

Exercice :

Concevoir un programme qui, à l'aide d'un switch :

- Déclare une variable meteo, égale à « froid »
- Selon la valeur de meteo :
 - Affiche « Sortez les maillots » si meteo est égale à chaud
 - Affiche « Préparez les bonnets » si meteo est égale à froid
 - Sinon, affiche : « Aucune idée du temps qu'il fera »



Les boucles

- For

```
int[] valeurs = {10, 14, 15, 20, 19};  
for (int i = 0; i < valeurs.length; i++) {  
    System.out.println(valeurs[i]);  
}
```

- Foreach

```
int[] valeurs = {10, 14, 15, 20, 19};  
for (int f : valeurs) {  
    System.out.println(f);  
}
```

Exercice :

Concevoir un programme qui :

- Déclare un tableau avec 2 valeurs : 11 et 12
- Afficher les 2 valeurs à l'aide d'un foreach



Les boucles

- While

```
int i = 1;
while (i < 3){
    System.out.println("Compteur : " + i);
    i++;
}
```

Exercice :

- Tester le code



Les boucles

- Do while

```
do {  
    instruction  
} while (condition);
```

Exercice :

- Avec un do while : faire un code qui demande le prénom de l'utilisateur tant que celui-ci n'en a pas entré un
- Si le prénom est entré, affichez le

Exercice :

- Refaire l'exercice avec un while() simple



Les fonctions et procédures

```
public void afficherSalutation(String prenom) {  
    System.out.println("Bonjour "+prenom+" !");  
}
```

```
public String saisirPrenom() {  
    Scanner s = new Scanner(System.in);  
    System.out.println("Quel est votre prénom ?");  
    String prenom = s.nextLine();  
    s.close();  
    return prenom;  
}
```

```
public static void main(String[] args) {  
    String prenom = saisirPrenom();  
    afficherSalutation(prenom);  
}
```

Exercice :

Concevoir un programme qui :

- Déclare une variable « age » égale à 30
- Passe la variable a une fonction privée « vieillir() »

La variable age doit vieillir d'un an après son passage par la fonction. Affichez la à la fin du programme.

Les exceptions

```
// .....
```

```
try {  
    val = s.nextInt();  
} catch (Exception e){  
    system.err.println("Veuillez  
    entrer un entier")  
}
```

Exercice :

Concevoir un programme qui :

- Demande à l'utilisateur son age
- Si une phrase est entrée au lieu d'un entier, afficher le message d'erreur ("Nous voulons un entier").
- Sinon, afficher : "Tu as XX ans"
- Faire en sorte que le programme s'arrêter s'il affiche un message d'erreur grâce à `System.exit(0)`



Les assertions

Il faut activer les assertions :

- Windows -> preferences
- Taper "Installed JREs"
- Sélectionner la première JRE (la seule normalement) et faire "edit"
- Dans "Default VM arguments", écrire : -ea (avec le tiret)



Les assertions

Une assertion permet de vérifier une condition considérée comme vraie.

Si cette condition est vraie, alors l'assertion sera muette.
Si elle est fausse, alors une erreur sera produite.

Une assertion s'introduit via le mot clé **assert** et elle est suivie de la condition à vérifier et éventuellement d'un message d'erreur.



Les assertions

```
package module1;

import java.util.Scanner;

public class Exceptions {

    public static void main(String[] args) {

        System.out.println("Quel temps fait-il cet été ?");
        Scanner sc = new Scanner(System.in);

        int temperature = sc.nextInt();
        assert temperature > 0 : "La température ne peut pas
être négative";

        System.out.println("Il fait " + temperature + "degrés");

        sc.close();

    }

}
```

TP



2.

L'utilisation de classes de Java



Idée générale de la Programmation Orientée Objet

- Regrouper des variables qui « vont bien ensemble »
 - Exemple :

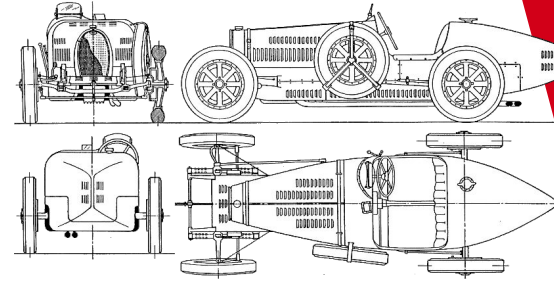
les variables heures, minutes et secondes dans une classe Temps

- Associer les fonctions et les procédures qui manipulent ces valeurs
 - Exemple :

les sous-algorithmes `convertirEnSecondes()` et `changerFuseauHoraire()` dans la classe Temps

Idée générale de la Programmation Orientée Objet

- Classe :
 - Un moule qui définit des attributs et des fonctions
- Instance :
 - Un élément que l'on construit grâce au moule



Pour faire simple



Des instances

Une classe



Création d'une variable de type GregorianCalendar

```
package module2;

import java.util.GregorianCalendar;

public class Test {

    public static void main(String[] args) {
        // déclaration d'une variable de type GregorianCalendar
        GregorianCalendar calendar;
    }
}
```



Utilisation d'un constructeur de la classe `GregorianCalendar`

```
package module2;

import java.util.GregorianCalendar;

public class Test {

    public static void main(String[] args) {
        // déclaration d'une variable de type
        GregorianCalendar
        GregorianCalendar calendar = new
        GregorianCalendar(1789, 7, 14);
    }

}
```

Exo ensemble :

- Créer un nouveau package "module2"
- Ecrire le code de gauche
- Découverte de la Javadoc via la classe `GregorianCalendar`
- Tester la fonction `getWeeksInWeekYear`

TP



3.

La création de classes

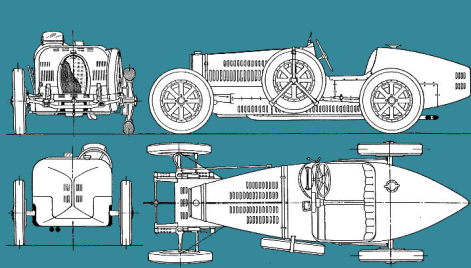


Déclaration d'une classe

- Une classe est définie dans un fichier portant le même nom que cette classe
- Le nom d'une classe commence par une majuscule
- Une classe est définie dans un package

Les attributs d'instance

- Ensemble de valeurs caractérisant une instance d'une classe



Bugatti35
-couleur:String
-roueSecours:boolean

Classe



Instance1 : Bugatti35	
couleur	"Bleu"
roueSecours	false



Instance2 : Bugatti35	
couleur	"Bordeau"
roueSecours	true

Instances

Les attributs d'instance

```
package module2;

/**
 * Classe modélisant un dé à jouer
 */
public class De {

    private int nbFaces;
    private int faceTiree;

}
```

De
-nbFaces:int -faceTiree:int

A faire : doter la classe De
d'attributs nbFaces et
faceTiree



Les visibilités au sein de la classe

Visibilité	Mot clef	Signification
Privée	private	Accessible uniquement au sein de la classe
Publique	public	Accessible de n'importe où
Package	Ø	Accessible uniquement au sein de ce package
Protégé	protected	Accessible au sein de la classe et de ses héritières



Le principe d'encapsulation

- Une classe est responsable de ses données
 - De l'extérieur de la classe, il est impossible de manipuler directement les attributs
 - Par contre, il est possible d'appeler l'une des méthodes qui elle a accès aux attributs



Les méthodes d'instance

```
private int nbFaces;  
private int faceTiree;  
private static Random rand = new Random();  
public int getNbFaces() {  
    return this.nbFaces;  
}  
public void setNbFaces(int nbFaces) {  
    this.nbFaces = nbFaces;  
}  
public int lancer() {  
    return this.faceTiree = De.rand.nextInt(this.nbFaces) + 1;  
}  
public int getFaceTiree() {  
    return this.faceTiree;  
}
```

A faire :

Insérez ce code dans votre classe
De

Déclaration d'une classe

```
package module3;

public class TestDe {

    public static void main(String[] args) {
        De monDe = new De();
        monDe.setNbFaces(6);
        do {
            System.out.println("Le dé a fait un " +
monDe.Lancer());
        } while (monDe.getFaceTiree() != 6);
    }

}
```

A faire :
Testez le programme



Le dé a fait un 4
Le dé a fait un 1
Le dé a fait un 2
Le dé a fait un 6

Les méthodes d'instance

```
package fr.eni.ecole.jeuDeDes;
```

```
public class TestDe {
```

```
    public static void main(String[] args) {
```

```
        De monDe = new De();
```

```
        monDe.nbFaces = 6;
```

```
        do {
```

```
            System.out.println("Le dé a fait un " + monDe.Lancer());
```

```
        } while (monDe.getFaceTiree() != 6);
```

```
    }
```

```
}
```

Interdit en raison de sa visibilité privée



Le constructeur par défaut

- Est présent lorsqu'il n'y a aucun constructeur de défini dans une classe
- Ne prend aucun argument
- Est public

Le constructeur par défaut

```
package fr.eni.ecole.jeuDeDes;
```

```
public class TestDe {
```

```
    public static void main(String[] args) {
```

```
        De monDe = new De();
```

```
        monDe.setNbFaces(6);
```

```
        do {
```

```
            System.out.println("Le dé a fait un " + monDe.Lancer());
```

```
        } while (monDe.getFaceTiree() != 6);
```

```
    }
```

```
}
```

Appel au constructeur par défaut car aucun constructeur n'est défini dans la classe De



Le dé a fait un 4
Le dé a fait un 1
Le dé a fait un 2
Le dé a fait un 6

Déclaration d'une classe

- Porte le même nom que la classe
- N'a pas de type de retour

```
public class De {  
  
    private int nbFaces;  
    private int faceTiree;  
    private static Random rand = new Random();  
  
    public De(int nbFaces) {  
        this.setNbFaces(nbFaces);  
        this.lancer();  
    }  
    ...  
}
```

A faire :

Renseigner le constructeur de la classe De

Le constructeur

```
package fr.eni.ecole.jeuDeDes;
```

```
public class TestDe {
```

```
    public static void main(String[] args) {
```

```
        De monDe = new De();
```

```
        monDe.setNbFaces(6);
```

```
        do {
```

```
            System.out.println("Le dé a fait un " + monDe.Lancer());
```

```
        } while (monDe.getFaceTiree() != 6);
```

```
    }
```

```
}
```

Dès qu'un constructeur est défini,
il n'y a plus de constructeur par défaut



Static

Le mot-clé static devant une variable (ou méthode) indique que celle-ci n'appartient pas à une instance particulière de la classe.

Les variables ou méthodes statiques appartiennent à la classe elle-même. On peut ainsi les utiliser sans avoir une instance créée.



Static

```
public class TestStatic {  
  
    private static String type = "humain";  
    private int age = 30;  
  
    public static String quiEstCe(){  
        if (type.equals("alien")){  
            return "un extra terrestre !";  
        }  
        if (type.equals("humain")){  
            return "ouf, un humain !";  
        }  
        return "aucune idée de l'espèce";  
    }  
  
    public int getAge(){  
        return this.age;  
    }  
}
```

Créez cette classe pour
comprendre le mot clé static



Static

```
public class ProgStatic{  
    public static void main (String[] args){  
        System.out.println(TestStatic.getAge());  
        System.out.println(TestStatic.quiEstCe());  
    }  
}
```

Créez cette classe pour comprendre le mot clé static et exécutez là.

La phrase “ouf, un humain s’affiche”.

Ajoutez ce code. Que remarquez vous ?

Exercice : sans modifier la fonction `getAge()`, faites en sorte que le programme affiche l’âge.

Les attributs de classe

- Défini avec le mot clef **static**

```
public class Terrain {  
    ...  
    private int coutConstruction;  
    private int niveauConstruction;  
  
    private static int nbMaisonsDispo = 32;  
    private static int nbHotelsDispo = 12;  
    ...  
}
```

Attributs d'instance

Attributs de classe

Les attributs de classe

Terrain	
-nbMaisonsDispo:int	29
-nbHotelsDispo:int	12

Attributs de classe

belleville : Terrain	
coutConstruction	50
niveauConstruction	2

bourse : Terrain	
coutConstruction	150
niveauConstruction	0

Attributs d'instance

lecourbe : Terrain	
coutConstruction	50
niveauConstruction	0

vaugirard : Terrain	
coutConstruction	50
niveauConstruction	0

paix : Terrain	
coutConstruction	200
niveauConstruction	1

Les méthodes de classe

- Méthode dont l'exécution ne dépend pas d'une instance particulière
- Définie avec le mot clef **static**
- Une méthode de classe ne peut accéder qu'aux attributs de classe (pas aux attributs d'instance)

```
private static void verifNbFaces(int nbFaces) throws Exception {  
    if (nbFaces <= 1)  
        throw new Exception("Un dé doit avoir au moins deux faces");  
}
```

Méthode de classe

```
public void setNbFaces(int nbFaces) throws Exception {  
    De.verifNbFaces(nbFaces);  
    this.nbFaces = nbFaces;  
}
```

Méthode d'instance



Les méthodes d'instance et méthodes de classe

	Méthode d'instance	Méthode de classe
Mot clef	Ø	static
Accès aux attributs de classe	<input type="checkbox"/>	<input type="checkbox"/>
Accès aux attributs d'instance	<input type="checkbox"/>	<input type="checkbox"/>
Appel depuis la classe	this .nomMethodeInstance() Exemple : this .lancer()	nomClasse.nomMethodeClasse() Exemple : De.verifNbFaces(6)
Appel hors de la classe	nomInstance .nomMethodeInstance() Exemple : de1 .lancer()	

TP

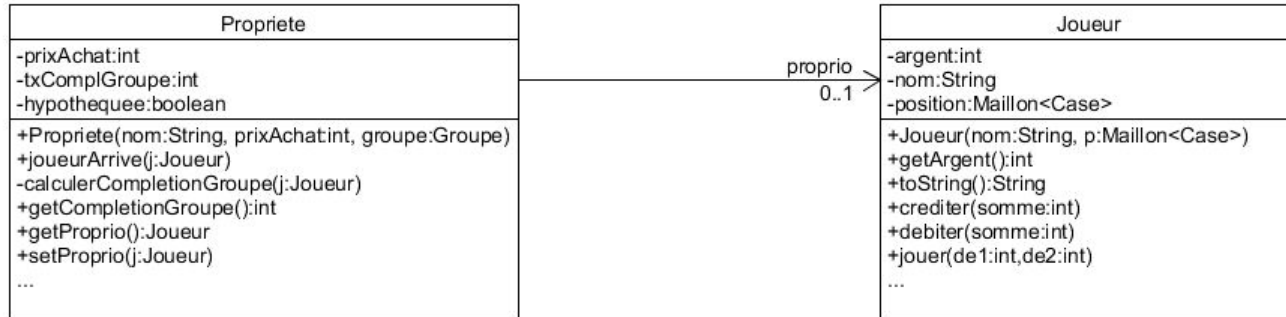


4.

Les associations

Les associations unidirectionnelles

Dans un Monopoly, par exemple..





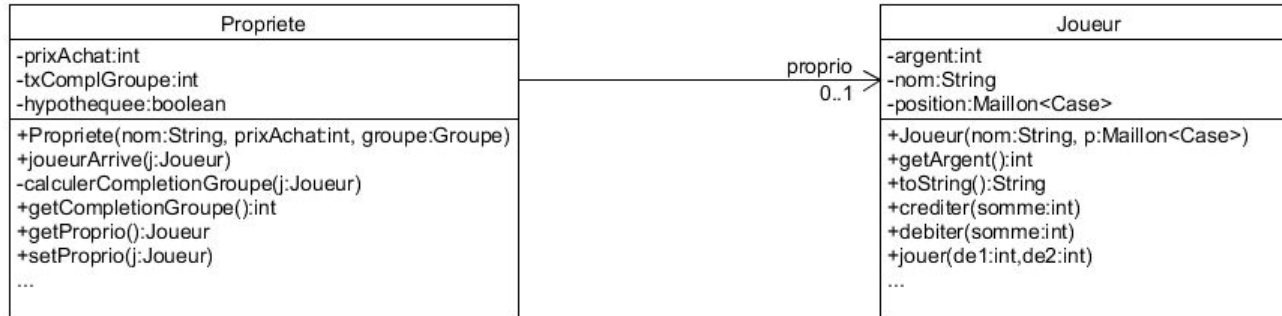
Les associations unidirectionnelles

```
public class Propriete {  
  
    private int prixAchat;  
    private int txComplGroupe;  
    private boolean hypotheeue;  
    private Joueur proprio;  
  
    public Joueur getProprio() {  
        return proprio;  
    }  
  
    public void setProprio(Joueur j) {  
        Joueur ancienProprio = this.proprio;  
        this.proprio = j;  
        if(ancienProprio != null)  
            this.calculerCompletionGroupe(ancienProprio);  
        this.calculerCompletionGroupe(j);  
    }  
    ...  
}
```

```
public class Joueur {  
  
    private int argent;  
    private String nom;  
    ...  
}
```

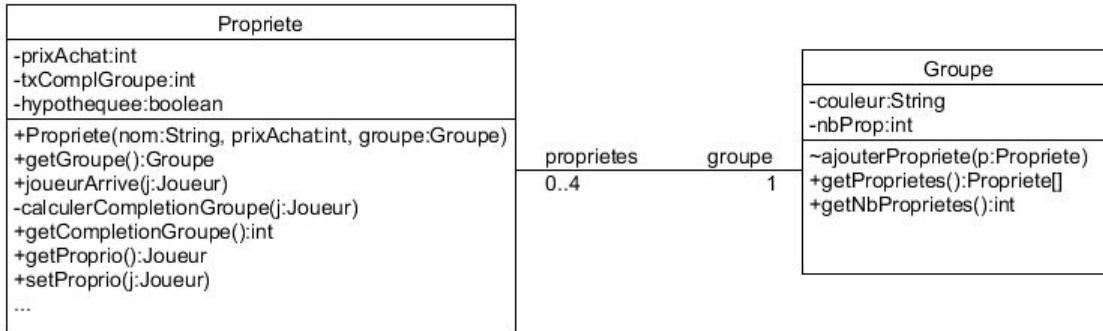
Les associations unidirectionnelles

- Une association unidirectionnelle n'est navigable que dans un seul sens
 - Dans l'exemple précédent :
 - Une propriété sait quel est son propriétaire
 - Un joueur ne connaît pas les propriétés qu'il possède



Les associations bidirectionnelles

- La navigation est possible dans les deux sens
 - Exemple :
 - Une propriété sait à quel groupe de propriétés elle appartient
 - Un groupe de propriété sait quelles sont les propriétés qui la compose





Les associations bidirectionnelles

```
public class Propriete {
    private int prixAchat;
    private int txComplGroupe;
    private boolean hypotheeue;

    private Joueur proprio;
    private Groupe groupe;

    public Propriete(int prixAchat, Groupe groupe) {
        this.groupe = groupe;
        groupe.ajouterPropriete(this);
        this.prixAchat = prixAchat;
        this.txComplGroupe = 0;
        this.hypotheeue = false;
    }

    public Groupe getGroupe() {
        return groupe;
    }
    ...
}
```

```
public class Groupe {
    private Propriete[] proprietes
        = new Propriete[4];
    private int nbProp = 0;

    void ajouterPropriete(Propriete p) {
        if(this.equals(p.getGroupe())) {
            this.proprietes[this.nbProp]=p;
            this.nbProp++;
        }
    }

    public Propriete[] getProprietes() {
        return this.proprietes;
    }

    public int getNbProprietes() {
        return this.nbProp;
    }
}
```

Les associations bidirectionnelles

belleville : Propriete	
prixAchat	60
txComplGroupe	0
Hypothèque	false
groupe	

mauve : Groupe	
nbProp	2
proprietes	

lecourbe : Propriete	
prixAchat	60
txComplGroupe	0
Hypothèque	false
groupe	

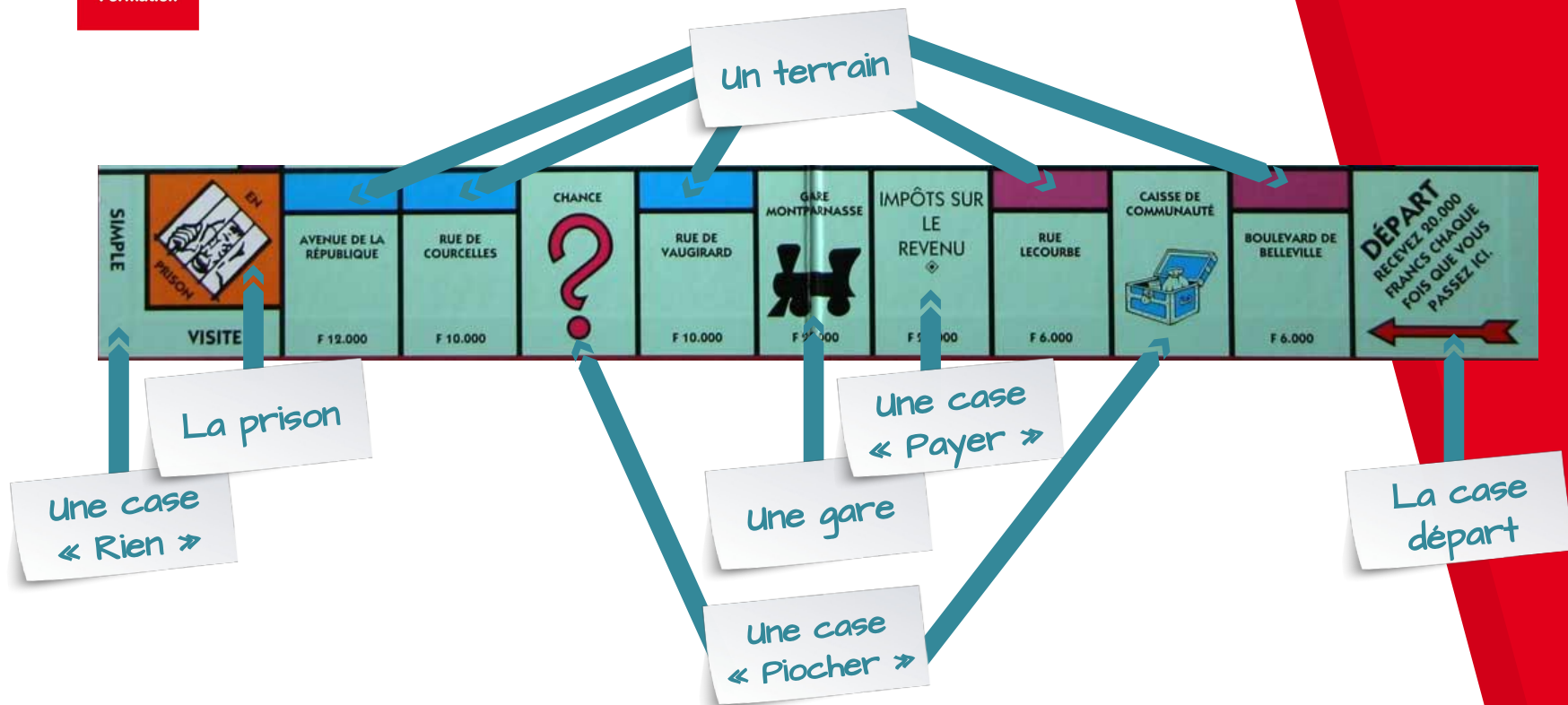
Série d'exos



5.

L'héritage

Les cases du plateau de jeu du Monopoly





Des classes avec du code dupliqué

```
public class Gare{  
  
    private String nom;  
    private int prix;  
    private int nbTotal;  
  
    public boolean  
    joueurPart(nom) {  
        // ...  
    }  
  
    // etc  
}
```

```
public class Impots{  
  
    private String nom;  
    private int prix;  
  
    public boolean  
    joueurPart(nom) {  
        // ...  
    }  
  
    // etc  
}
```

```
public class Terrain{  
  
    private String nom;  
    private int prix;  
    private int nbTotal;  
    private String couleur;  
  
    public boolean  
    joueurPart(nom) {  
        // ...  
    }  
  
    // etc  
}
```



La solution : mutualisation du code par héritage

```
public class Case{  
  
    private String nom;  
    private int prix;  
    private int nbDetenue;  
  
    public boolean  
    joueurPart(nom) {  
        // ...  
    }  
  
    // etc  
}
```

```
public class Gare extends Case{  
  
    // les attributs et méthodes  
    propres à Gare  
  
}  
  
public class ImpotTaxe extends  
Case{  
  
    // les attributs et méthodes  
    propres à ImpotTaxe  
  
}
```

```
public class Terrain extends  
Case{  
  
    // les attributs et méthodes  
    propres à Terrain  
  
}
```

La classe parent

```
public class UneCase{  
    protected String nom;  
    protected int prix;  
    protected int nbDetenue;  
  
    public Case(String nom) {  
        this.nom = nom;  
    }  
    public boolean joueurPart(nom) {  
        System.out.println(nom + " est parti de la case " + this.nom);  
        return true;  
    }  
}
```

Visibilité
protected

Une classe enfant

`extends` est le mot clef permettant d'indiquer l'héritage

```
public class Terrain extends UneCase{
```

```
    private String couleur;
```

```
    public Terrain(String nom, String couleur) {
```

```
        super(nom);
```

```
        this.couleur = couleur;
```

```
    }
```

```
    @Override
```

```
    public void joueurPart(nom) {
```

```
        super.joueurPart(nom);
```

```
        System.out.println("Il était sur une case " + this.couleur);
```

```
    }
```

```
}
```

Ajout d'un attribut supplémentaire en plus de celui défini dans Case

L'annotation `@Override` indique la substitution de la méthode

La méthode `joueurArrive()` est substituée : la version de cette méthode définie dans une classe parent est remplacée par celle-ci

`super()` permet de faire appel à l'un des constructeurs de la classe parent



On teste !

```
public class UneCase {  
    protected String nom;  
    protected int prix;  
    protected int nbDetenue;  
  
    public UneCase(String nom) {  
        this.nom = nom;  
    }  
    public boolean joueurPart(String nom) {  
        System.out.println(nom + " est parti de la case " +  
this.nom);  
        return true;  
    }  
}
```

Exercice :

Créer ces deux classes + un fichier MonopolyTest.java où vousinstancierez une UneCase et une Terrain. Les deux instances lanceront la fonction "joueurPart"

```
public class Terrain extends UneCase {  
  
    private String couleur;  
  
    public Terrain(String nom, String  
couleur) {  
        super(nom);  
        this.couleur = couleur;  
    }  
  
    @Override  
    public boolean joueurPart(String nom) {  
        super.joueurPart(nom);  
        System.out.println("Il était sur une  
case " + this.couleur);  
  
        return true;  
    }  
}
```



toString

- Utilisée pour savoir quelle classe détient quels attributs ainsi que les valeurs des instances.
- Méthode déjà existante, overriding obligatoire.

Premier test : faites appel à la méthode `toString()` à partir de l'instance de votre choix dans votre programme `main`.

Second test : overridez la méthode `toString` dans la classe de l'instance concernée pour afficher



Polymorphisme

Ce que l'on vient de faire (overrider une méthode d'une classe parent) porte un nom : le polymorphisme.

```
public class Animal {  
  
    public void crier() {  
        System.out.println("un cri d'animal");  
    }  
}
```

```
public class Chien extends Animal {  
  
    @Override  
    public void crier() {  
        System.out.println("Whouaf whouaf !");  
    }  
}
```

```
public class Chat extends Animal {  
  
    @Override  
    public void crier() {  
        System.out.println("Miaou !");  
    }  
}
```

```
Animal animal = new Animal();  
animal.crier(); // affiche "un cri d'animal"
```

```
Chat chat = new Chat();  
chat.crier(); // affiche "Miaou !"
```

```
Chien chien = new Chien();  
chien.crier(); // affiche "Whouaf whouaf !"
```

TP



6.

Les classes abstraites et les interfaces



L'abstraction

On commence le module avec un exercice fil rouge : package exo6



Les classes abstraites

```
public class Animal {  
  
    private String nom;  
    protected int nbCroc;  
    ...  
}
```

```
Chat chat = new Chat("Felix", 20);  
Chien chien = new Chien("Bill", 30);
```

Comment faire pour empêcher un développeur de faire :

```
Animal vache = new Animal("Margueritte", 10);
```



Les classes abstraites

En rajoutant abstract

```
public abstract class Animal {  
  
    private String nom;  
    protected int nbCroc;  
    ...  
}
```

Quand est-ce que j'utilise abstract alors ?



Les classes abstraites

A la suite de l'exo 6 :

Le code fonctionne et cela nous permet de créer une propriété.

Toutefois, cette classe étant une classe parent, elle ne sert qu'à regrouper les attributs de ses classes filles (ici Gare et Terrain) et ne devrait jamais être instanciée.

Faites en sorte d'empêcher l'instanciation de Propriété.



Les interfaces

Une interface, c'est comme une liste de course dans laquelle on liste les fonctions obligatoires d'une classe plutôt que des ingrédients à acheter.

Démonstration

Exo 6.2

TP



7.

Les énumérations



Les énumérations

- Dans votre code, package module7 :
 - Créez une classe Electricite avec un nom (le nom d'un courant électrique) et un niveau de criticité (faible, normale ou haute).
 - Ensuite, créez un courant électrique dans votre main.



Les énumérations

Comment faire pour limiter le niveau de criticité à “faible”, “normal” ou “haute” et pas à autre chose ?

Il existe un type particulier en Java qui permet de fournir implémentation : l'énumération.



Les énumérations

```
public class Electricite {  
  
    // Déclaration de la liste  
    public enum Criticite {  
        FAIBLE, NORMAL, HAUTE  
    }  
  
    // Déclaration de l'attribut  
    private Criticite niveauCriticite;  
  
    ...  
  
}
```

```
Dans le main : Electricite courant = new  
Electricite("Courant alternatif", Criticite.FAIBLE);  
System.out.println(courant.getNiveauCriticite());
```



Les énumérations

Quand utiliser une énumération ?

Dès lors qu'un attribut d'une instance peut avoir un nombre limité de valeur, on doit créer une énumération pour obliger le développeur à n'utiliser que les valeurs permises.



8.

La généricité



Découverte des collections

```
public class Animal{  
  
    public static void main(String[] args) {  
        List<String> animaux = new ArrayList<>();  
  
        list.add("Vache");  
        list.add(1);  
  
        System.out.println(list.get(0));  
    }  
}
```

- Une collection est un tableau sans taille prédéfinie !
- Un bug apparaît : on ne peut ajouter un entier dans la liste !



Classe générique

- Vocabulaire :

On dit que `ArrayList` est une classe générique car on peut choisir ce qu'on va lui passer :

`ArrayList<String>` ou `ArrayList<Int>` ou, pourquoi pas, un type qu'on aura créé nous même : `ArrayList<Animal>`



Vers les méthodes génériques

L'utilisation de la généricité devient utile lorsque l'on crée des méthodes génériques. Avant d'en créer une, réalisons un programme.

Créez une classe Vehicule avec un attribut private int vitesse; et un String nom
Puis deux classes filles (Voiture et Moto) qui étendent vehicule.

Dans le main, instanciez deux voitures et une moto



Vers les méthodes génériques

Affichez le véhicule le + rapide à la suite, voici l'algo:

- si la vitesse de voiture1 > la vitesse de voiture 2
 - si la vitesse de voiture1 > la vitesse de moto1
 - Afficher "Le plus rapide est "+voiture1.getNom()
 - sinon
 - Afficher "Le plus rapide est "+moto1.getNom()
- sinon si la vitesse de voiture2 > la vitesse de moto1
 - Afficher "Le plus rapide est "+voiture2.getNom()
- sinon
 - Afficher "Le plus rapide est "+moto1.getNom()

Ajoutez maintenant une moto 2, une moto3 et une voiture3 puis modifiez votre code.

Ou pas ...



Vers les méthodes génériques

Faisons une méthode `getPlusRapide` qui va trier tout ça. A mettre dans la classe `Vehicule`.

```
public static Vehicule getPlusRapide(List<Vehicule> vehicules) {  
    Vehicule plusRapide = null;  
    int vitesse = 0;  
    for (Vehicule vehicule : vehicules) {  
        if(vehicule.getVitesse() >= vitesse)  
        {  
            vitesse = vehicule.getVitesse();  
            plusRapide = vehicule;  
        }  
    }  
    return plusRapide;  
}
```



Vers les méthodes génériques

La méthode `getPlusRapide` attend une `arrayListe` de véhicule.
Dans votre main, construisez la et passez la à la fonction `getPlusRapide`.
Essayez d'afficher le nom du véhicule le plus rapide (c'est pas facile).

```
List<Vehicule> vehicules = new ArrayList<>();
```

```
vehicules.add(voiture1);  
vehicules.add(voiture2);  
vehicules.add(moto1);
```

```
Vehicule plusRapide = Vehicule.getPlusRapide(vehicules);
```

```
System.out.println(plusRapide.getNom());
```



Vers les méthodes génériques

Si l'on doit ajouter de nouveaux véhicules à la liste, il faudrait faire un `.add()` à chaque fois.. un peu lourd.

Démo

```
protected static ArrayList<Animal> animaux = new ArrayList<>();
```

```
animaux.add(this); <- dans le constructeur
```

```
// Générer les getters et setters
```



Les énumérations

Parfait. Grâce à la classe générique `ArrayList`, si on ajoute une voiture 3, 4, 5 ... il suffira juste de les ajouter à la liste, tout simplement !

Imaginons maintenant que j'ai besoin de connaître la liste des VOITURES les plus rapides.

A la suite de votre main, déclarez une liste de voitures, ajoutez les 2 voitures dedans et appelez `Vehicule.getVoiturePlusRapide` pour afficher la voiture la plus rapide.



La généricité

Quelle est l'une, si ce n'est LA règle d'or quand on veut faire un bon code ?

LE MOINS POSSIBLE
DE CODE QUI SE
REPETE (svp)



Créer une méthode générique

Dans notre code, qu'est-ce qui se répète ?

Oui, quelle bonne réponse ! Il s'agit effectivement de la méthode `getPlusRapide` et `getVoiturePlusRapide`

La seule différence, c'est que l'une prend une liste de véhicules et renvoie un véhicule tandis que l'autre prend une liste de voitures et renvoie une voiture



Créer une méthode générique

La méthode générique à l'aide.

Dans le main, remplacez l'appel à `getVoiturePlusRapide` par `getPlusRapide`.
Dans `Vehicule`, supprimez la méthode `getVoiturePlusRapide`

Objectif : rendre `getPlusRapide` générique pour qu'elle accepte aussi bien les `Vehicule`, les voitures, les motos,....

TP