

Module M15

La programmation orientée objet JAVA

Chapitre 2

Programmation Orientée Objets avec **JAVA**

Programmation Orientée Objets avec JAVA

Plan du chapitre

1. Un peu d'histoire
2. Présentation de la POO
3. Concept N°1 : **classe et objet**

Un peu
d'histoire

Un peu d'histoire

Le paradigme de la POO

- **Apparu** au début des années 1960 par les Norvégiens *Ole-Johan Dahl et Kristen Nygaard* et **poursuivi** par les travaux **d'Alan Kay** dans les années 1970
 - Consiste en la **définition des briques logiciels** et **leur interaction**
 - Ces briques logiciel constituent **les objets (objects)**.
-
- **Un objet** peut représenter **un concept, une idée, ou toute entité du monde physique**, comme une voiture, une personne ou encore une page d'un livre.

Un peu d'histoire

Le paradigme de la POO

- Un objet possède une structure interne et des comportements.
- Un objet peut communiquer avec ses pairs dans son environnement.
- Les objets peuvent interagir entre eux pour répondre à un objectif complexe.
- L'interaction entre les objets est assurée par des relations qui peuvent exister entre les différents objets.

La conception des objets et leurs relations est une phase primordiale pour mieux résoudre un problème.

Un peu d'histoire

Le paradigme de la POO

- La programmation orientée objet a été adoptée par le langage Smalltalk
- À partir de 1995, JAVA est présenté au public en tant que langage de programmation orientée objets

Présentation de la POO

La programmation orientée objet

La méthode Orientée Objet

Il s'agit de **manipuler des types personnalisés**

Ces types peuvent être:

- **Prédéfinis** dans les bibliothèques du langage Orienté Objet
- Ou **définis** par le développeur

Le type personnalisé représente un modèle « **Classe** dans la suite du cours » qui donne lieu à des **objets**

Les objets et leurs interactions dans l'environnement d'exécution représentent le résultat « réalisation du besoin » attendu par le programme

La programmation orientée objet

Les principaux concepts de la Programmation Orientée Objet:

1. Classe et Objet
2. Encapsulation (Accessibilité)
3. Héritage
4. Polymorphisme

Concept N° 1

Classe et Objet

Classe et Objet

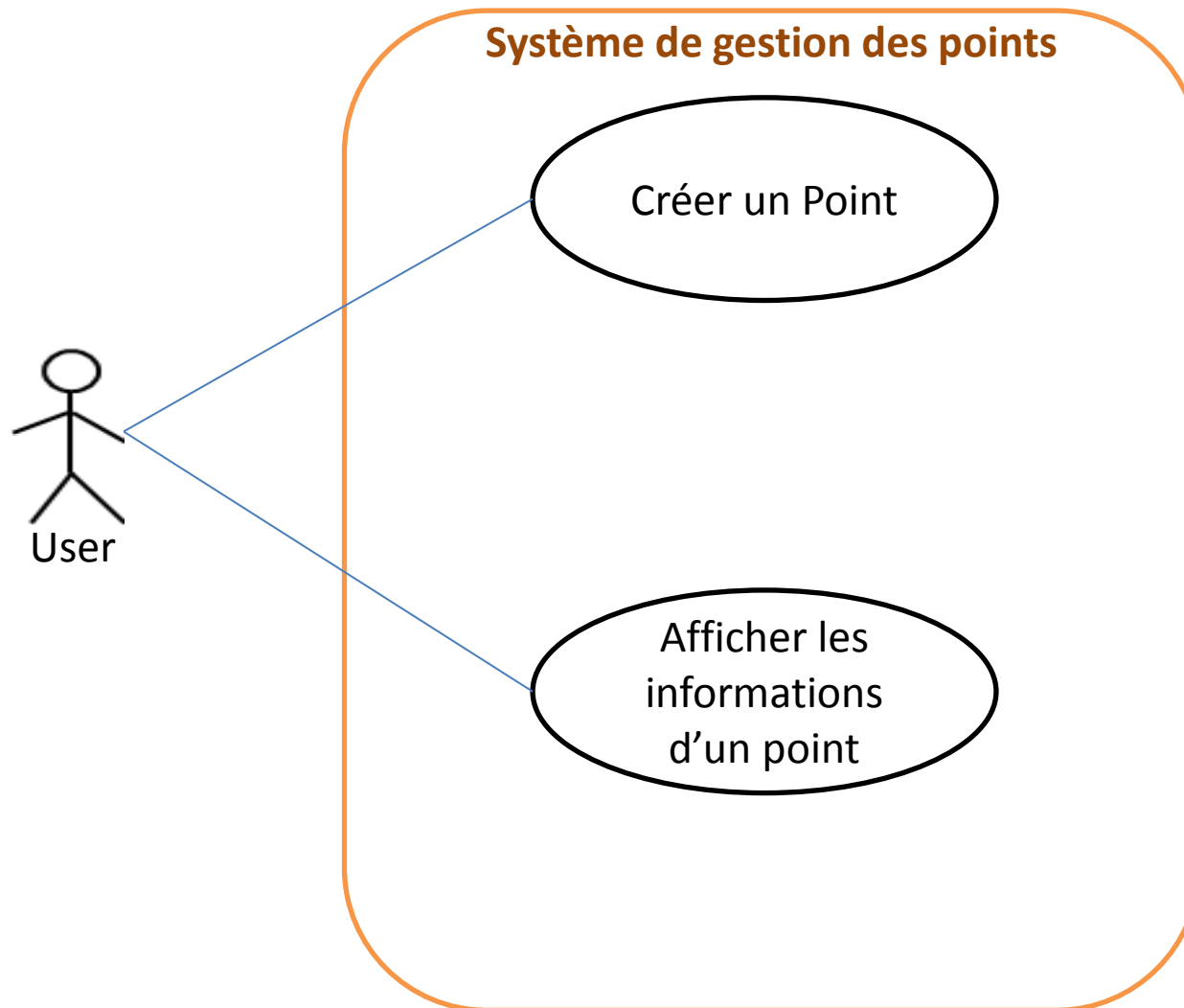
Exemple d'introduction

L'exemple que nous allons traiter contient les éléments suivants:

- une classe « **Point** » qui va servir comme un type complexe et personnalisé et dont le code source est dans le fichier Point.java
- Une classe « **Program** » qui va servir comme programme principal et dont le code source est dans le fichier Program.java
- Les deux classes appartiennent au même package : [com.step1](#)

Expression des besoins du projet

Diagramme de cas d'utilisation



Classe et Objet

Dans cette partie, nous allons traiter les concepts suivants:

- Classe
- Attributs
- Constructeur & Objet
- package
- Méthodes
- Objet courant
- Constructeur avec paramètres
- Membres statiques

NB: Durant cette partie, nous nous limitons à l'usage de la visibilité ***public*** des attributs

Classe et Objet

Dans cette partie, nous allons traiter les concepts suivants:

- Classe
- Attributs
- Constructeur & Objet
- package
- Méthodes
- Objet courant
- Constructeur avec paramètres
- Membres statiques

NB: Durant cette partie, nous nous limitons à l'usage de la visibilité **public** des attributs

Classe: Qu'est ce que c'est?

- Une classe représente un **type** personnalisé et complexe.
- Le code source d'une classe appartient à un fichier dont l'extension est « **.java** »
- Une classe a des membres:
 - Les attributs
 - Les méthodes
- Les membres doivent avoir une visibilité
 - **private**, **public**, **protected**, par défaut
- Une classe appartient à un package

Les éléments d'une classe

Compartiment

Désignation

Exemple

Compartiment 1

Nom de la classe

Point

Compartiment 2

Les attributs

abscisse
ordonne
couleur

Compartiment 3

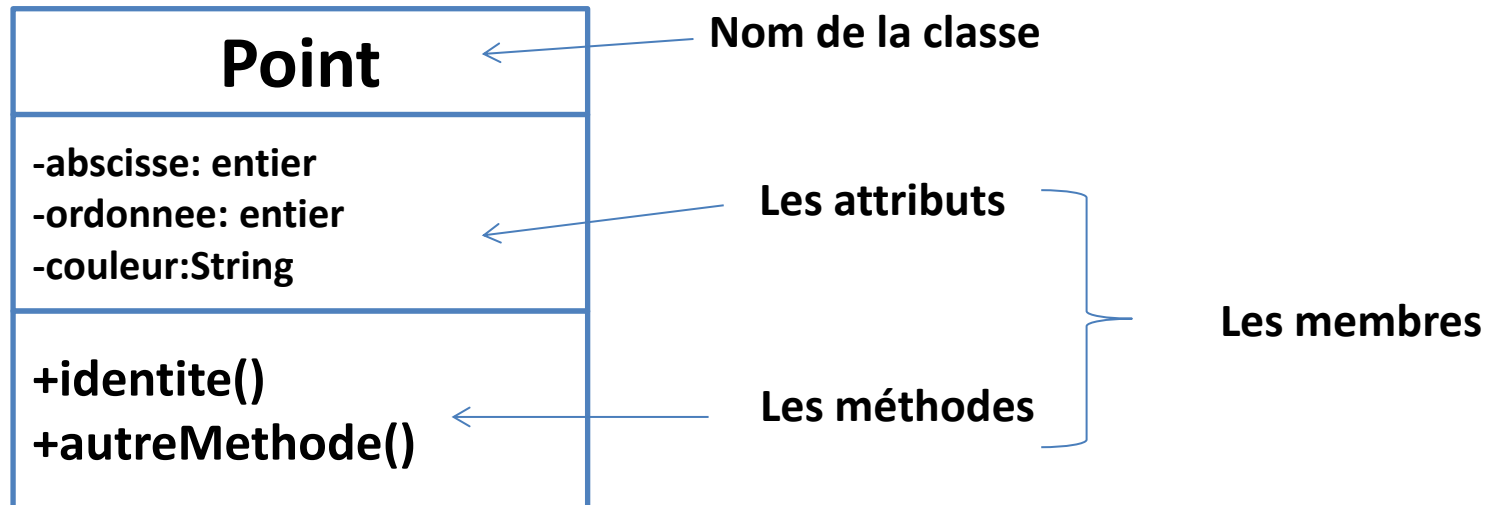
Les méthodes

identite()

Les classes et leurs relations sont conçues dans la phase de conception.

(CF diagramme de classes en UML)

Représentation UML d'une classe



Le nom de la classe

Le nom de la classe est obligatoire. Par convention il commence avec une lettre majuscule.

Exemple

```
class Point{// début de la classe  
  
} //fin de la classe
```

Classe et Objet

Dans cette partie, nous allons traiter les concepts suivants:

- Classe
- Attributs
- Constructeur & Objet
- package
- Méthodes
- Objet courant
- Constructeur avec paramètres
- Membres statiques

NB: Durant cette partie, nous nous limitons à l'usage de la visibilité **public** des attributs

Les attributs de la classe

- Les attributs d'une classe constituent les caractéristiques des objets générés à partir de cette classe
- Un attribut est caractérisé par :
 - un nom
 - un type
 - **une visibilité (portée)**
 - Éventuellement une valeur initiale

Exemple

```
class Point{  
public double abscisse;  
public double ordonnee;  
public String couleur="noir";  
} //fin de la classe
```

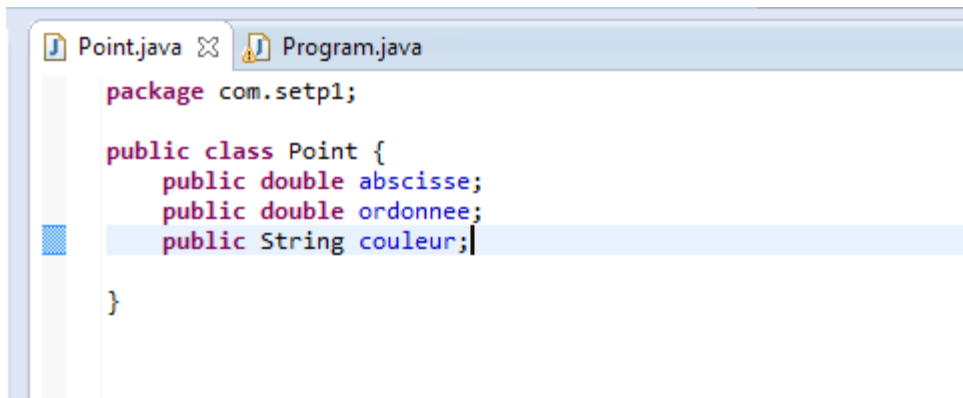
Pour le moment, nous acceptons le modificateur public représente la visibilité des attributs

CF. Encapsulation des attributs

Une classe et ses attributs

La classe « Point » décrivant un type complexe

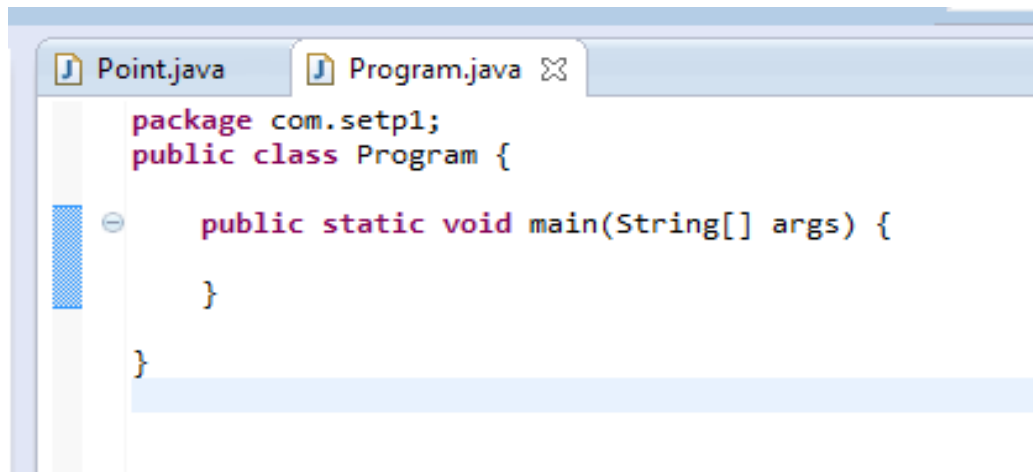
```
package com.setp1;  
public class Point  
{  
    public double abscisse;// attribut 1  
    public double ordonnee; //attribut 2  
    public string couleur= "noir";//attribut3;  
}
```



Une autre classe de tests

Une classe « Program » représentant le programme principal et contenant la méthode main

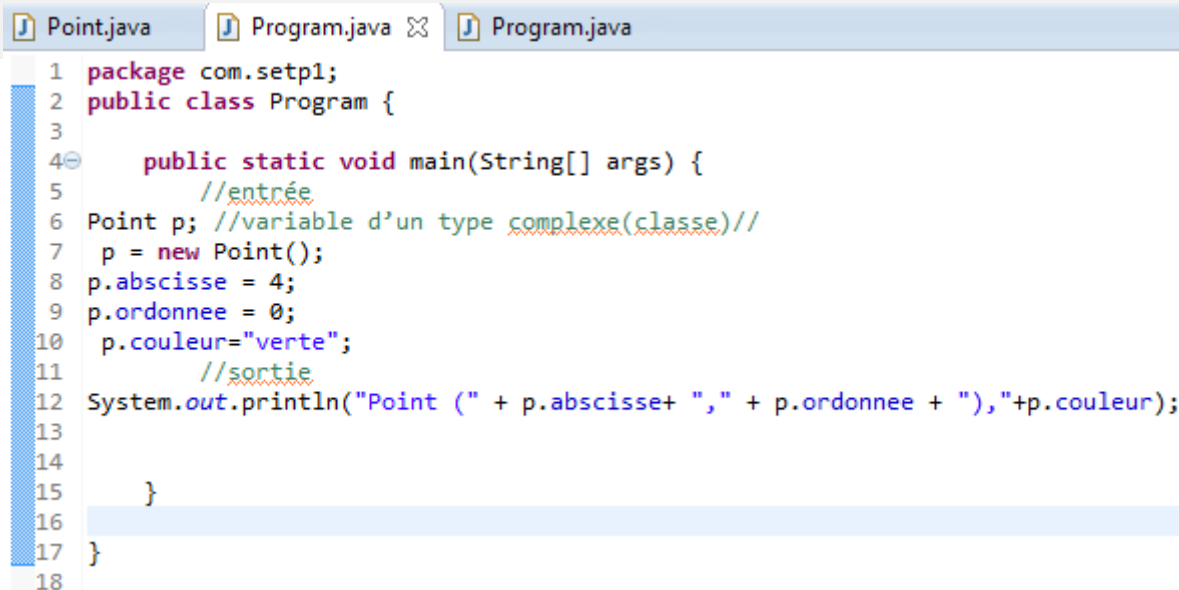
```
package com.setp1;  
public class Program {  
  
    public static void main(String[] args) {  
  
    }  
}
```



Classe et Objet

Un programme principal utilisant la classe « Point » Vs Programme simple

```
package com.setp1;
public class Program {
    public static void main(String[] args) {
        Point p; //variable d'un type complexe(classe)//
        p = new Point();
        p.abcisse = 4;
        p.ordonnee = 0;
        p.couleur="verte";
        System.out.println("Point (" + p.abcisse+ "," + p.ordonnee + "),"+p.couleur);
    }
}
```

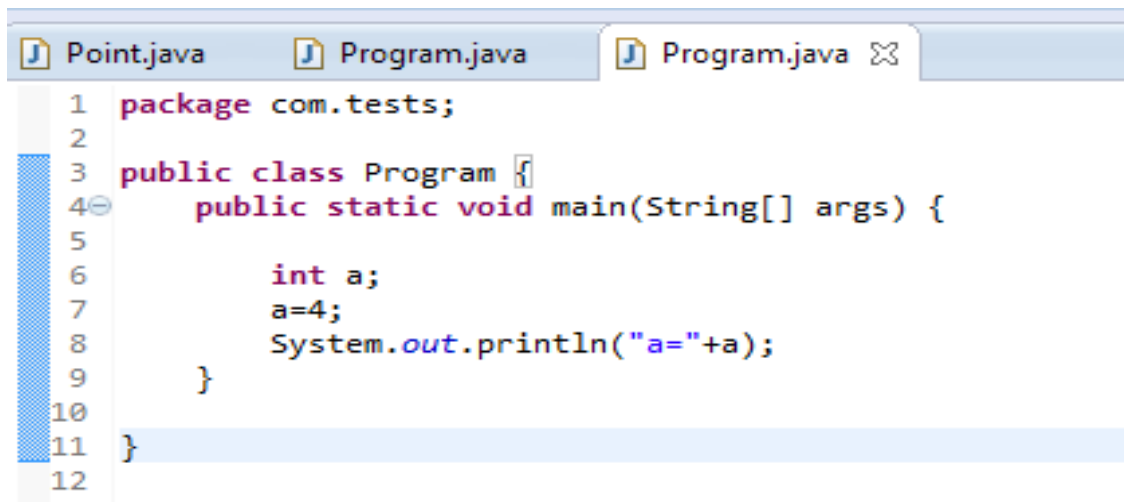


```
1 package com.setp1;
2 public class Program {
3
4     public static void main(String[] args) {
5         //entrée
6         Point p; //variable d'un type complexe(classe)//
7         p = new Point();
8         p.abcisse = 4;
9         p.ordonnee = 0;
10        p.couleur="verte";
11        //sortie
12        System.out.println("Point (" + p.abcisse+ "," + p.ordonnee + "),"+p.couleur);
13
14
15    }
16
17 }
18
```


Classe et Objet

Un programme principal utilisant la classe « Point » Vs Programme simple

```
package com.tests;  
public class Program{  
    public static void main(string[] args){  
        //entrée  
int a;  
    a=3;  
    Console.WriteLine("a=" + a); //ligne 3  
}  
}
```

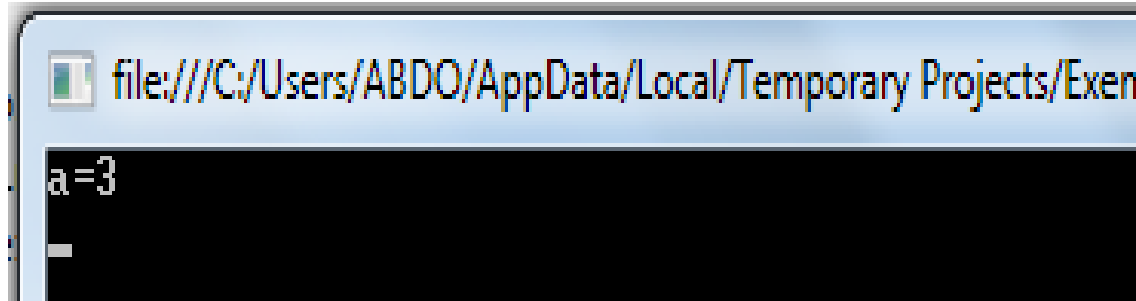


```
1 package com.tests;  
2  
3 public class Program {  
4     public static void main(String[] args) {  
5  
6         int a;  
7         a=4;  
8         System.out.println("a="+a);  
9     }  
10  
11 }  
12
```

Classe et Objet

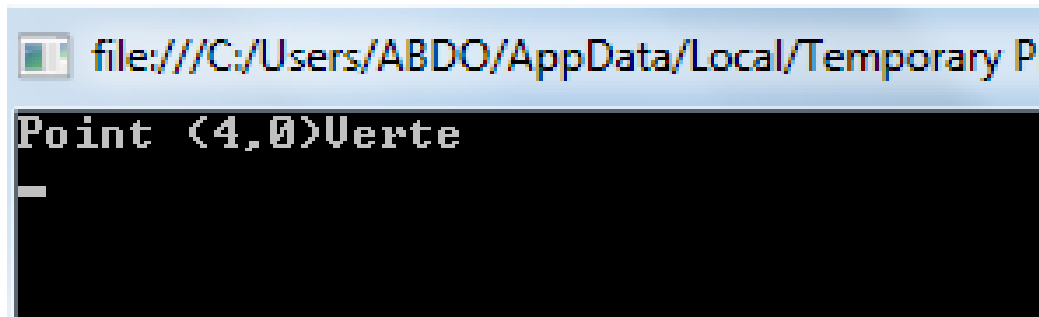
Type primitif (simple) vs type classe (complexe)

Le résultat du programme de la variable de type primitif



```
file:///C:/Users/ABDO/AppData/Local/Temporary Projects/Exen
a=3
_
```

Le résultat du programme utilisant la classe Point



```
file:///C:/Users/ABDO/AppData/Local/Temporary P
Point <4,0>Verte
_
```

Classe et Objet

Type primitif (simple) vs type classe (complexe)

Analyse de l'exemple de gestion de la variable de type primitif (int)

- Ligne 6: création de la variable a et allocation de l'espace mémoire nécessaire pour stocker l'entier a
- Ligne 7: affecter la valeur 3 à la variable a
- ligne 8 : affichage de la valeur de a



Utilisation directe des types primitifs

Classe et Objet

Type primitif (simple) vs type classe (complexe)

Analyse de l'exemple de gestion de la variable de type complexe et personnalisé (Point)

- Ligne 6: création d'une **référence** de type Point
- Ligne 7: Allocation de l'espace mémoire pour stocker le Point P à l'aide du **constructeur: Point()**
- Ligne 8,9,10: affecter des valeurs aux variables (attributs) primitifs appartenant à la classe
- ligne 11: affichage des valeurs des variables du Point p

Classe et Objet

Dans cette partie, nous allons traiter les concepts suivants:

- Classe
- Attributs
- Constructeur & Objet
- package
- Méthodes
- Objet courant
- Constructeur avec paramètres
- Membres statiques

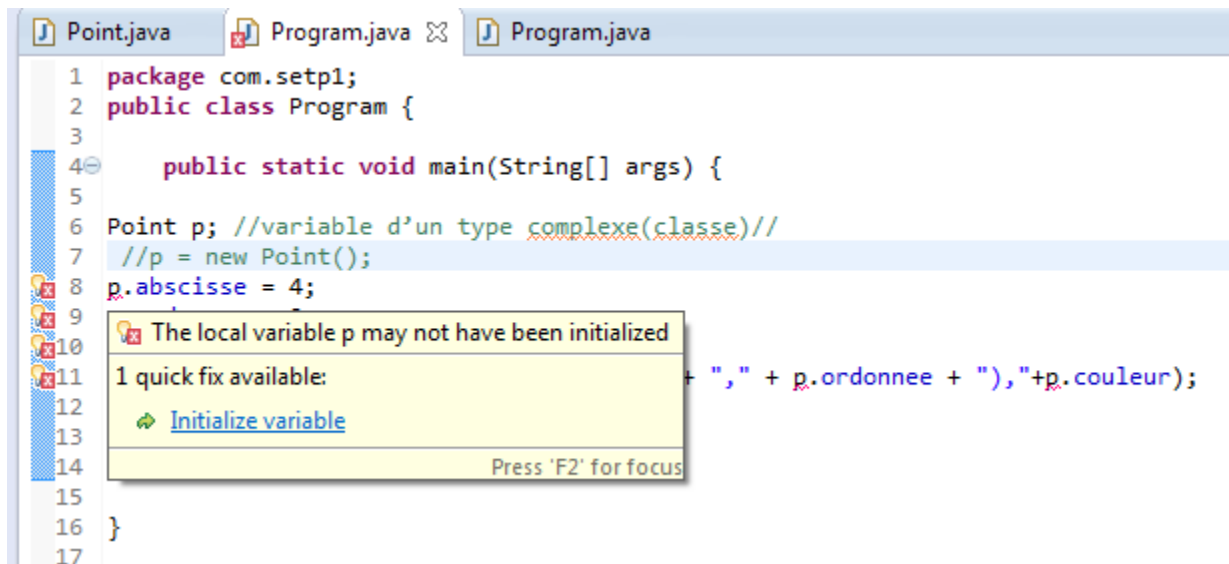
NB: Durant cette partie, nous nous limitons à l'usage de la visibilité **public** des attributs

Constructeur

Le constructeur: à quoi sert?

Tests et conclusions

Mettre en commentaire la ligne 7, puis compiler et exécuter



```
1 package com.setp1;
2 public class Program {
3
4     public static void main(String[] args) {
5
6         Point p; //variable d'un type complexe(classe)//
7         //p = new Point();
8         p.abscisse = 4;
9
10        //p.ordonnee = 5;
11        //p.couleur = "rouge";
12        //System.out.println("Point: " + p.abscisse + "," + p.ordonnee + ")," + p.couleur);
13
14    }
15
16 }
17
```

The local variable p may not have been initialized

1 quick fix available:

- [Initialize variable](#)

Press 'F2' for focus

L'IDE eclipse signale une erreur à la première utilisation du point p.

Conclusion: L'instruction `p=new Point()` permet d'allouer une zone mémoire pour un point en retournant l'adresse de cette zone. L'adresse sera affectée à la référence p.

- Cette instruction permet de créer une instance ou un objet de type Point
- On ne peut pas utiliser directement une référence d'un objet mais il faut qu'elle pointe sur une zone mémoire récupérée à travers la commande `new Point();`

Constructeur

Le constructeur et son rôle

- Le constructeur d'une classe permet de créer des objets à l'aide de l'opérateur **new**
- Le constructeur d'une classe permet d'initialiser les valeurs des attributs de l'objet
- Le constructeur représente une méthode particulière
(CF section des méthodes)
- Un constructeur peut être surchargé
(CF section de surcharge des méthodes)
- Le constructeur par défaut ne contient aucun paramètre. Il peut exister d'une manière implicite.

Exemple: usage d'un constructeur par défaut

```
public static void main(string[] args) {  
    Point p;// à ce niveau, nous pouvons pas utiliser l'objet p  
    p=new Point();// maintenant, l'objet p est utilisable  
}
```

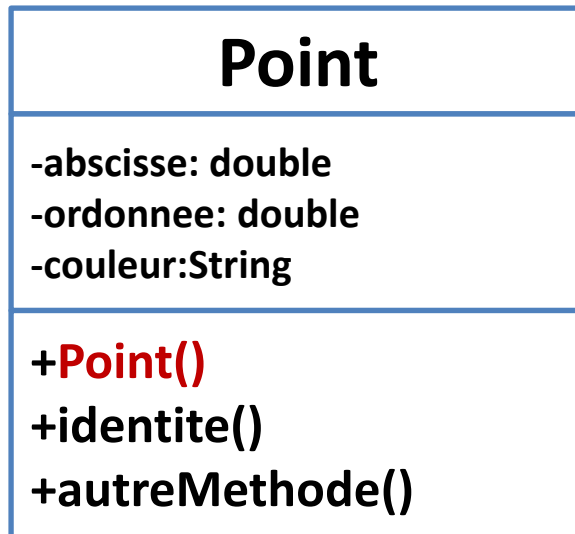
Constructeur

Classe – Objet - Constructeur

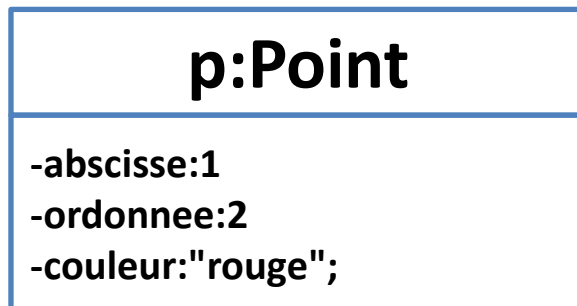
- La relation entre la classe et l'objet est une relation d'instanciation
- L'instance ou l'objet est la variable de type Maclasse (ici Point)
- Un constructeur permet de créer un objet ou une instance. On dit aussi, il permet d'instancier une classe

Classe et Objet: représentation UML

Classe – Objet – Constructeur: Représentation UML



Classe Point



Objet p

L'objet

Objet

- Un objet est une structure informatique définie par un état et un comportement
- $\text{Objet} = \text{état} + \text{comportement}$
- L'état regroupe les valeurs instantanées de tous les attributs de l'objet.
- Le comportement regroupe toutes les compétences et décrit les actions et les réactions de l'objet. Autrement dit le comportement est défini par les opérations que l'objet peut effectuer.
- L'état d'un objet peut changer dans le temps.
- Généralement, c'est le comportement qui modifie l'état de l'objet

Classe et Objet

Dans cette partie, nous allons traiter les concepts suivants:

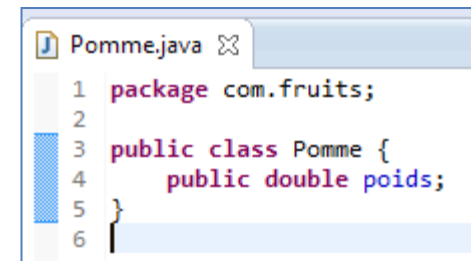
- Classe
- Attributs
- Constructeur & Objet
- package
- Méthodes
- Objet courant
- Constructeur avec paramètres
- Membres statiques

NB: Durant cette partie, nous nous limitons à l'usage de la visibilité **public** des attributs

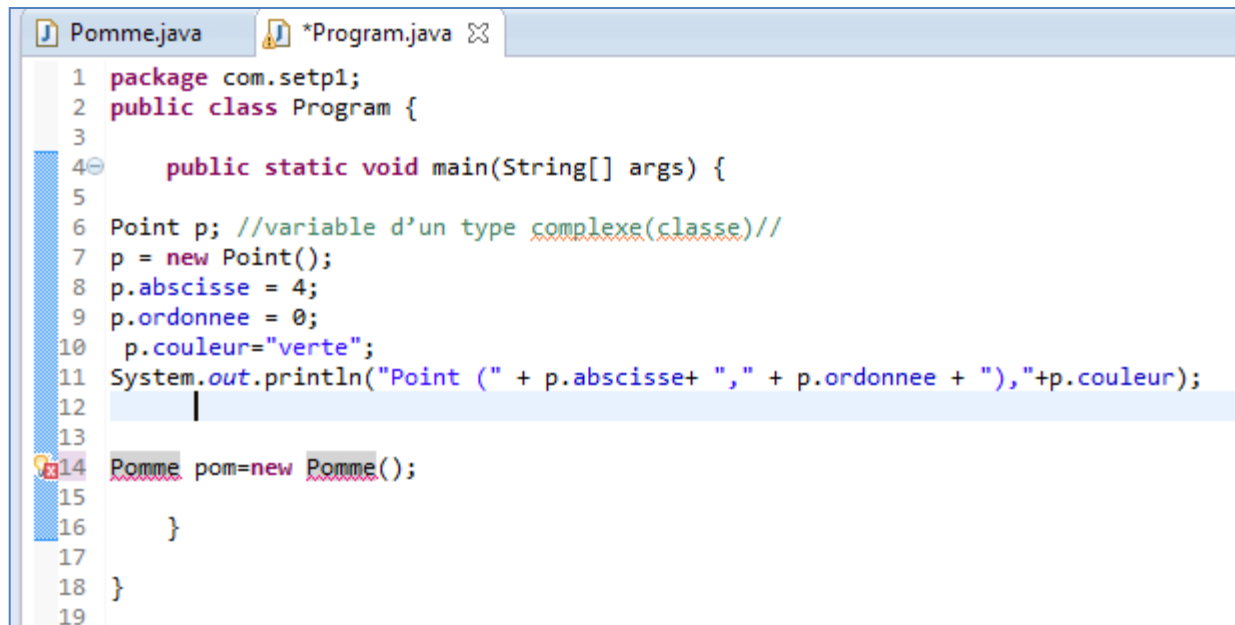
Package

Tests et conclusions

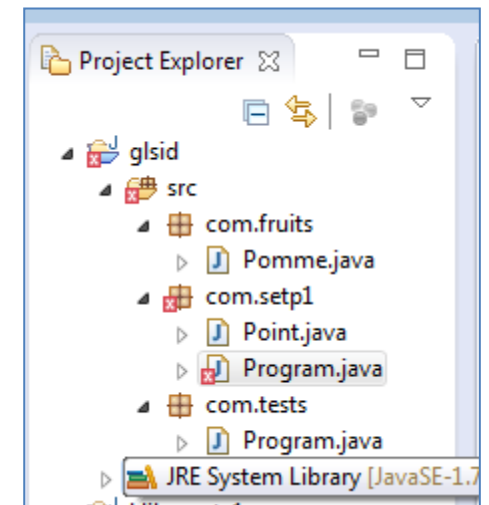
- Créer un package com.fruits;
- Créer une classe pomme dans ce package



```
Pomme.java
1 package com.fruits;
2
3 public class Pomme {
4     public double poids;
5 }
6
```



```
Pomme.java *Program.java
1 package com.setp1;
2 public class Program {
3
4     public static void main(String[] args) {
5
6         Point p; //variable d'un type complexe(classe)//
7         p = new Point();
8         p.abscisse = 4;
9         p.ordonnee = 0;
10        p.couleur="verte";
11        System.out.println("Point (" + p.abscisse+ "," + p.ordonnee + "),"+p.couleur);
12
13
14        Pomme pom=new Pomme();
15
16    }
17
18 }
19
```

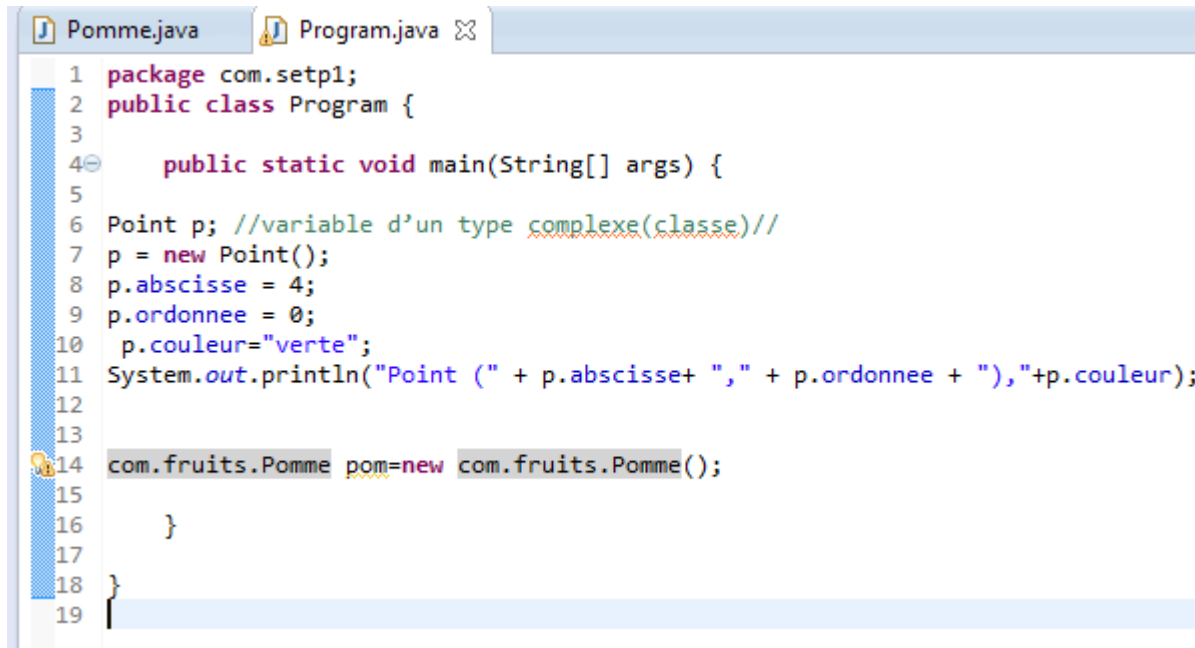


- Les classes Pomme et Point n'appartiennent pas au même package
- le type Pomme(la classe Pomme) n'est pas connue dans la classe Program

Package

Tests et conclusions

- Dans le fichier source de la classe Program, spécifier le chemin complet de la classe Pomme: `com.fruits.Pomme`



```
1 package com.setp1;
2 public class Program {
3
4     public static void main(String[] args) {
5
6         Point p; //variable d'un type complexe(classe)//
7         p = new Point();
8         p.abscisse = 4;
9         p.ordonnee = 0;
10        p.couleur="verte";
11        System.out.println("Point (" + p.abscisse+ "," + p.ordonnee + "),"+p.couleur);
12
13
14        com.fruits.Pomme pom=new com.fruits.Pomme();
15
16    }
17
18 }
19
```

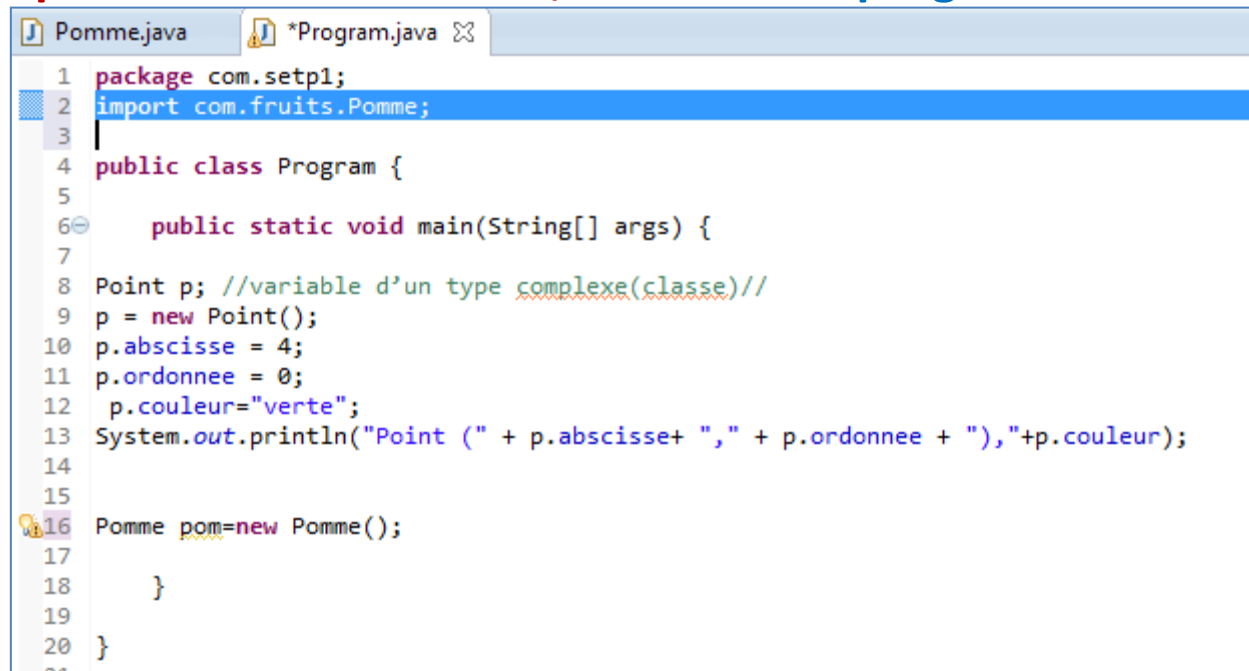
Méthode 1

- Spécifier le package de la classe à utiliser permet de la reconnaître
- Cette méthode, nous oblige à chaque utilisation de spécifier le chemin complet. Ceci complique l'écriture du code!!

Package

Tests et conclusions

- Dans le fichier source de la classe Program, ajouter la directive **import com.fruits.Pomme;** au début du programme



```
1 package com.setp1;
2 import com.fruits.Pomme;
3
4 public class Program {
5
6     public static void main(String[] args) {
7
8         Point p; //variable d'un type complexe(classe)//
9         p = new Point();
10        p.abcisse = 4;
11        p.ordonnee = 0;
12        p.couleur="verte";
13        System.out.println("Point (" + p.abcisse+ "," + p.ordonnee + "),"+p.couleur);
14
15
16        Pomme pom=new Pomme();
17
18    }
19
20 }
```

Méthode 2

- la classe Pomme est connue
- Cette méthode évite d'écrire à chaque fois le chemin complet

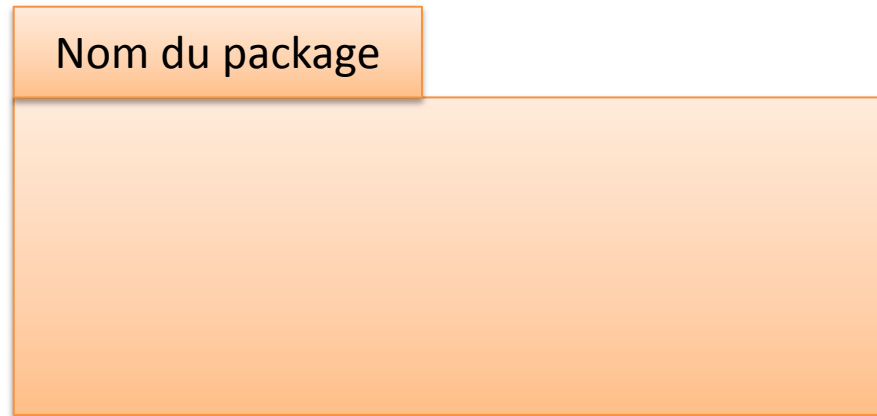
Package

- Une classe peut appartenir à un seule package
- En java, les packages permettent une structuration physique des éléments du code source
- Un package peut contenir plusieurs classes
- Le package permet une meilleure organisation des différents éléments du projet

**NB: Pour une bonne subdivision du projet en plusieurs packages,
voir la partie diagramme de package en UML**

Package

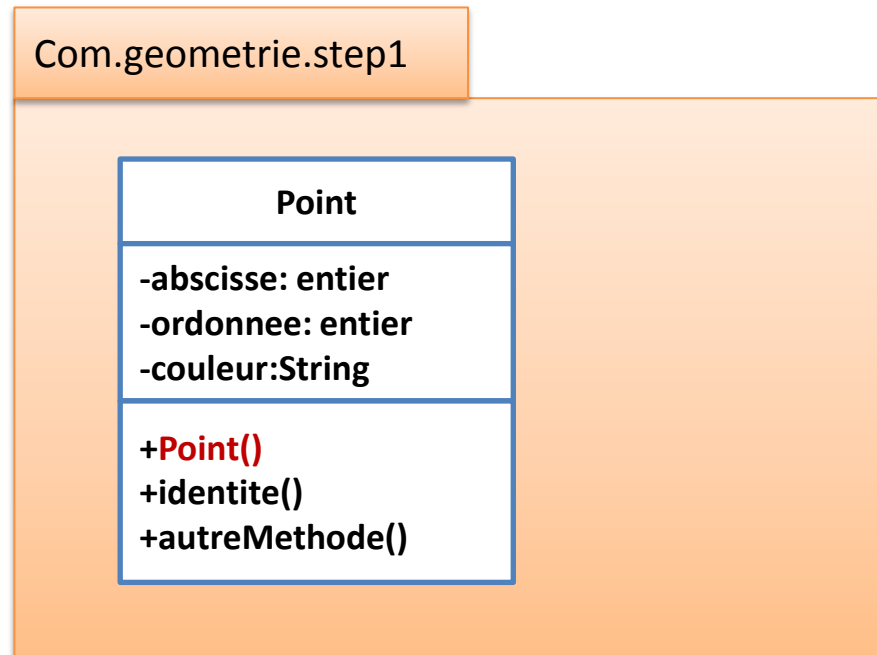
- Les packages offrent un mécanisme général pour la partition des modèles et le regroupement des éléments de la modélisation
- Chaque package est représenté graphiquement par un dossier
- Les packages divisent et organisent les modèles de la même manière que les dossier organisent le système de fichier



Représentation UML des packages

CF: diagramme de package. Ce diagramme vient après le diagramme de use case

Le package-Exemple



Représentation UML des packages

Classe et Objet

Dans cette partie, nous allons traiter les concepts suivants:

- Classe
- Attributs
- Constructeur & Objet
- package
- Méthodes
- Objet courant
- Constructeur avec paramètres
- Membres statiques

NB: Durant cette partie, nous nous limitons à l'usage de la visibilité ***public*** des attributs

Les méthodes

- Les méthodes ou les opérations d'une classe décrivent le comportement des objets générées à partir de cette classe.
- Les méthodes assurent l'interaction entre les différents objets à travers l'échange de messages
- Une méthode permet de réaliser une ou plusieurs tâches
- Une méthode représente une fonction ou une procédure qui permet d'exécuter plusieurs instructions
- Une méthode est caractérisée par:
 - Un nom
 - Une signature (type de retour et paramètres)
 - Une visibilité (portée)

Les méthodes

```
public class Point{  
  // les attributs  
  public void mon_etat()  
  {  
    System.out.println ("mon état:");  
    System.out.println("("+abscisse+ ","+ordonnee+") "+couleur );  
  }  
} //fin de la classe
```

```
1 package com.setp1;  
2  
3 public class Point {  
4     public double abscisse;  
5     public double ordonnee;  
6     public String couleur;  
7  
8     public void mon_etat()  
9     {  
10         System.out.println ("mon état:");  
11         System.out.println("("+abscisse+ ","+ordonnee+") "+couleur );  
12     }  
13 }  
14  
15
```

Les méthodes

Utilisation de la méthode

```
Program.java Point.java
1 package com.setp1;
2
3 public class Program {
4
5     public static void main(String[] args) {
6
7         Point p;
8         p = new Point();
9         p.abscisse = 4;
10        p.ordonnee = 0;
11        p.couleur="verte";
12        System.out.println("Accès au Point depuis l'extérieur de la classe");
13        System.out.println("Point (" + p.abscisse+ ", " + p.ordonnee + "),"+p.couleur);
14
15        System.out.println("Accès au Point depuis l'intérieur de la classe");
16        p.mon_etat();
17    }
18 }
19
20 }
21
```

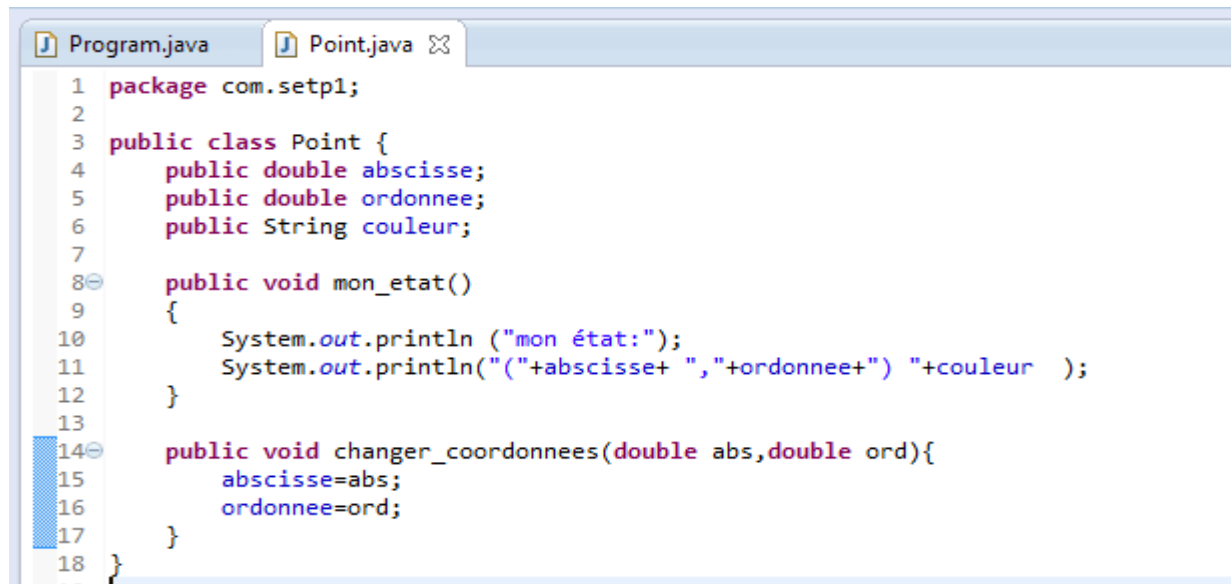
Résultat du programme

```
Markers Properties Servers Data Source Explorer Sni
<terminated> Program [Java Application] C:\Program Files\Java\jdk1.7.0_2
Accès au Point depuis l'extérieur de la classe
Point (4.0,0.0),verte
Accès au Point depuis l'intérieur de la classe
mon état:
(4.0,0.0) verte
```

Les méthodes

Ajouter une méthode pour changer l'état d'un objet

```
public class Point{  
    // les attributs  
    public void changer_coordonnees(double abs,double ord){  
        abscisse=abs;  
        ordonnee=ord;  
    }  
} //fin de la classe
```



```
1 package com.setp1;  
2  
3 public class Point {  
4     public double abscisse;  
5     public double ordonnee;  
6     public String couleur;  
7  
8     public void mon_etat()  
9     {  
10        System.out.println ("mon état:");  
11        System.out.println("(" +abscisse+ "," +ordonnee+" )"+couleur );  
12    }  
13  
14    public void changer_coordonnees(double abs,double ord){  
15        abscisse=abs;  
16        ordonnee=ord;  
17    }  
18 }
```

Les méthodes

Utilisation de la méthode `changer_coordonnees`

```
Program.java  Point.java
1  package com.setp1;
2
3  public class Program {
4
5      public static void main(String[] args) {
6
7      Point p;
8      p = new Point();
9      p.abscisse = 4;
10     p.ordonnee = 0;
11     p.couleur="verte";
12     System.out.println("avant changement de l'état");
13     p.mon_etat();
14     p.changer_coordonnees(5, 5);
15     System.out.println("après changement de l'état");
16     p.mon_etat();
17     }
18
19 }
20
```

Résultat du programme

```
Markers  Properties  Servers  Data Source Explorer
<terminated> Program [Java Application] C:\Program Files\Java\jdk
avant changement de l'état
mon état:
(4.0,0.0) verte
après changement de l'état
mon état:
(5.0,5.0) verte
```

Les méthodes

Surcharge des méthodes

- Soit la méthode `maj_etat ()` suivante qui permet de changer l'état de l'objet

```
public void maj_etat(){  
abscisse=0;  
ordonnee=0;  
couleur="noir";  
}
```

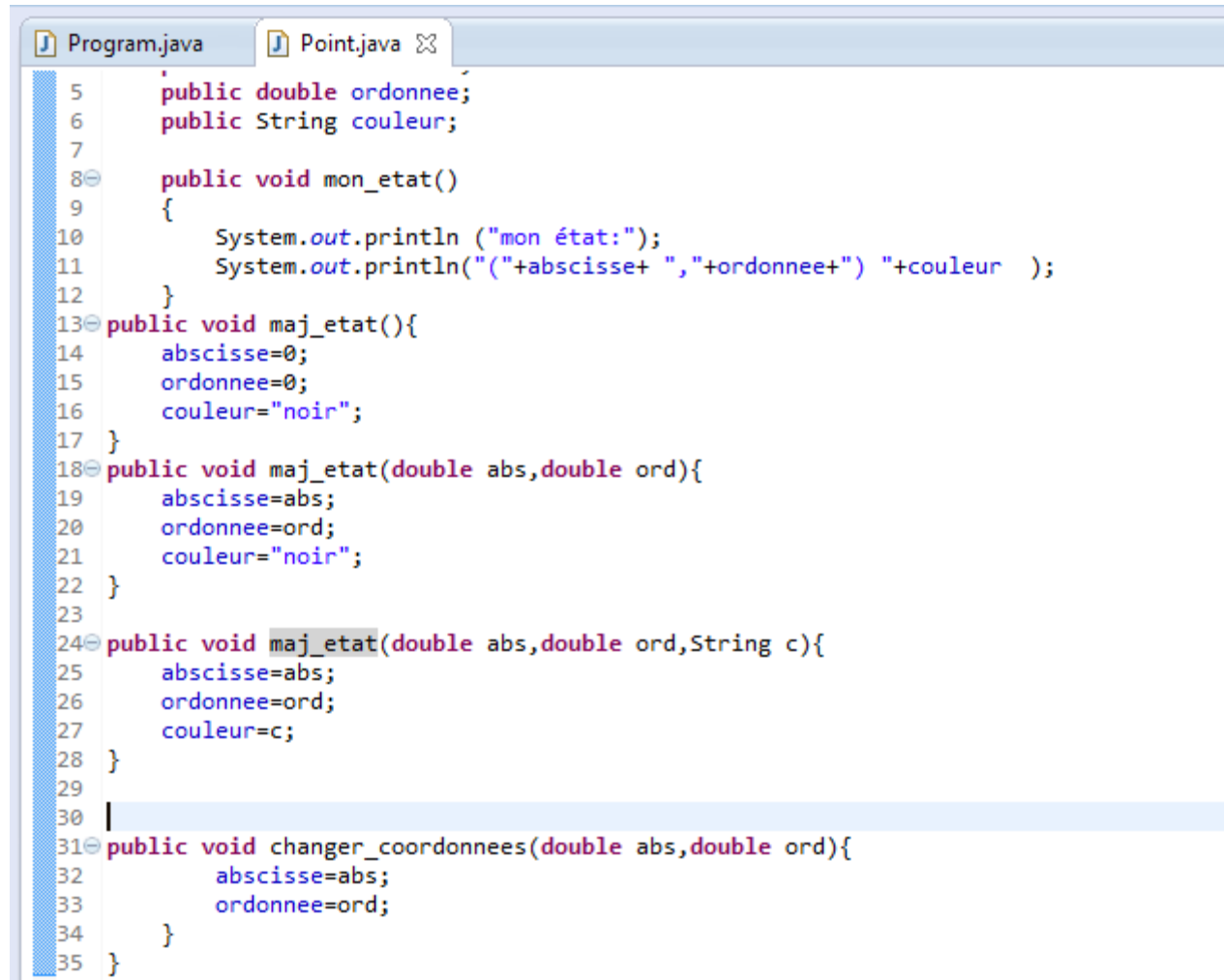
- Cette méthode peut être surchargée en changeant sa signature

```
public void maj_etat(double abs,double ord){  
abscisse=abs;  
ordonnee=ord;  
couleur="noir";  
}  
public void maj_etat(double abs,double ord,String c){  
abscisse=abs;  
ordonnee=ord;  
couleur=c;  
}
```

Le même nom avec des
signatures différentes
=
surcharge

Les méthodes

Surcharge de la méthode maj_etat



```
Program.java | Point.java ✕
5      public double ordonnee;
6      public String couleur;
7
8      public void mon_etat()
9      {
10         System.out.println ("mon état:");
11         System.out.println("(" + abscisse + ", " + ordonnee + ") " + couleur );
12     }
13     public void maj_etat(){
14         abscisse=0;
15         ordonnee=0;
16         couleur="noir";
17     }
18     public void maj_etat(double abs,double ord){
19         abscisse=abs;
20         ordonnee=ord;
21         couleur="noir";
22     }
23
24     public void maj_etat(double abs,double ord,String c){
25         abscisse=abs;
26         ordonnee=ord;
27         couleur=c;
28     }
29
30
31     public void changer_coordonnees(double abs,double ord){
32         abscisse=abs;
33         ordonnee=ord;
34     }
35 }
```

Les méthodes

Utilisation de la méthode `maj_etat`

Résultat du programme

```
avant changement de l'état
mon état:
(4.0,0.0) verte
après maj_etat()
mon état:
(0.0,0.0) noir
après maj_etat(8,8)
mon état:
(8.0,8.0) noir
après maj_etat(9,9,'rouge')
mon état:
(9.0,9.0) rouge
```

```
Program.java  Point.java
1 package com.setp1;
2
3 public class Program {
4     public static void main(String[] args) {
5         Point p;
6         p = new Point();
7         p.abscisse = 4;
8         p.ordonnee = 0;
9         p.couleur="verte";
10        System.out.println("avant changement de l'état");
11        p.mon_etat();
12        p.maj_etat();
13        System.out.println("après maj_etat()");
14        p.mon_etat();
15
16        p.maj_etat(8,8);
17        System.out.println("après maj_etat(8,8)");
18        p.mon_etat();
19
20        p.maj_etat(9,9,"rouge");
21        System.out.println("après maj_etat(9,9,'rouge')");
22        p.mon_etat();
23    }
24 }
```

Classe et Objet

Dans cette partie, nous allons traiter les concepts suivants:

- Classe
- Attributs
- Constructeur & Objet
- package
- Méthodes
- Objet courant
- Constructeur avec paramètres
- Membres statiques

NB: Durant cette partie, nous nous limitons à l'usage de la visibilité **public** des attributs

L'objet courant

- Dans une méthode, on appelle « objet courant », **l'objet sur lequel a été appelée la méthode**
- Un objet courant est représenté par la référence « **this** »

```
public void maj_etat(double abscisse,double  
ordonnee,String couleur){  
this.abscisse=abscisse;  
this.ordonnee=ordonnee;  
this.couleur=couleur;  
}
```

- l'objet sur lequel a été appelée la méthode est **p**
- La référence **p** et la référence « **this** » pointent sur la même zone mémoire.

```
*Program.java  Point.java  
1 package com.setp1;  
2  
3 public class Program {  
4     public static void main(String[] args) {  
5         Point p;  
6         p = new Point();  
7         p.abscisse = 4;  
8         p.ordonnee = 0;  
9         p.couleur="verte";  
10        System.out.println("avant changement de l'état");  
11        p.mon_etat();  
12        p.maj_etat();  
13        System.out.println("après maj_etat()");  
14        p.mon_etat();  
15  
16        p.maj_etat(8,8);  
17        System.out.println("après maj_etat(8,8)");  
18        p.mon_etat();  
19  
20        p.maj_etat(9,9,"rouge");//l'objet sur lequel a été appe  
21        System.out.println("après maj_etat(9,9,'rouge')");  
22        p.mon_etat();  
23    }  
24 }  
25
```

L'objet courant

Exemple

```
Program.java Point.java
3 public class Point {
4     public double abscisse;
5     public double ordonnee;
6     public String couleur;
7
8     public void mon_etat()
9     {
10         System.out.println ("mon état:");
11         System.out.println ("+"+abscisse+ " "+ordonnee+" "+couleur );
12     }
13     public void maj_etat(){
14         abscisse=0;
15         ordonnee=0;
16         couleur="noir";
17     }
18     public void maj_etat(double abscisse,double ordonnee){
19         this.abscisse=abscisse;
20         this.ordonnee=ordonnee;
21         this.couleur="noir"; //opérateur this pas obligatoire
22     }
23
24     public void maj_etat(double abscisse,double ordonnee,String couleur){
25         this.abscisse=abscisse;
26         this.ordonnee=ordonnee;
27         this.couleur=couleur;
28     }
29     public void changer_coordonnees(double abs,double ord){
30         abscisse=abs;
31         ordonnee=ord;
32     }
33 }
34
```

```
Program.java Point.java
1 package com.setp1;
2
3 public class Program {
4     public static void main(String[] args) {
5         Point p;
6         p = new Point();
7         p.abscisse = 4;
8         p.ordonnee = 0;
9         p.couleur="verte";
10        System.out.println("avant changement de l'état");
11        p.mon_etat(); //l'objet sur lequel a été appelé la méthode
12        p.maj_etat(); //l'objet sur lequel a été appelé la méthode
13        System.out.println("après maj_etat()");
14        p.mon_etat();
15
16        p.maj_etat(8,8); //l'objet sur lequel a été appelé la méthode
17        System.out.println("après maj_etat(8,8)");
18        p.mon_etat();
19
20        p.maj_etat(9,9,"rouge"); //l'objet sur lequel a été appelé
21        System.out.println("après maj_etat(9,9,'rouge')");
22        p.mon_etat();
23    }
24 }
25
```

Classe et Objet

Exercice d'application

- Écrit une méthode dans la classe point qui permet de calculer la distance entre deux points
- Utiliser la méthode sqrt de la bibliothèque Math
Math.sqrt(a): permet de calculer la racine carrée de a

Classe et Objet

Dans cette partie, nous allons traiter les concepts suivants:

- Classe
- Attributs
- Constructeur & Objet
- package
- Méthodes
- Objet courant
- Constructeur avec paramètres
- Membres statiques

NB: Durant cette partie, nous nous limitons à l'usage de la visibilité **public** des attributs

Le constructeur avec paramètres

- Comme pour les méthodes, un constructeur peut être surchargé
- Un constructeur peut avoir un ou plusieurs arguments pour initialiser l'objet. Ces arguments sont fournis pendant l'initialisation avec le mot clé new

```
Program.java  Point.java ✕  
1 package com.setp1;  
2  
3 public class Point {  
4     public double abscisse;  
5     public double ordonnee;  
6     public String couleur;  
7  
8     public Point(){  
9         this.abscisse=0;  
10        this.ordonnee=0;  
11        this.couleur="noir";  
12    }  
13    public Point(double abscisse, double ordonnee){  
14        this.abscisse=abscisse;  
15        this.ordonnee=ordonnee;  
16    }  
17    public Point(double abscisse, double ordonnee,String couleur){  
18        this.abscisse=abscisse;  
19        this.ordonnee=ordonnee;  
20        this.couleur=couleur;  
21    }  
22
```

```
Program.java ✕  Point.java  
1 package com.setp1;  
2  
3 public class Program {  
4     public static void main(String[] args) {  
5         Point p;  
6         p = new Point();  
7         Point p2=new Point();  
8         Point p3=new Point(3,3);  
9         Point p4=new Point(4,4,"Bleu");  
10    }
```

Compléter le programme pour afficher l'état de chaque objet

Le constructeur avec paramètres

Exemple

```
Program.java Point.java ✕
1 package com.setp1;
2
3 public class Point {
4     public double abscisse;
5     public double ordonnee;
6     public String couleur;
7
8     /* public Point(){
9         this.abscisse=0;
10        this.ordonnee=0;
11        this.couleur="noir";
12    } */
13    public Point(double abscisse, double ordonnee){
14        this.abscisse=abscisse;
15        this.ordonnee=ordonnee;
16    }
17    public Point(double abscisse, double ordonnee, String couleur){
18        this.abscisse=abscisse;
19        this.ordonnee=ordonnee;
20        this.couleur=couleur;
21    }
22 }
```

- Une fois un constructeur avec paramètres est mis en place, le constructeur sans paramètres par défaut n'est plus implicite.

```
Program.java ✕ Point.java
1 package com.setp1;
2
3 public class Program {
4     public static void main(String[] args) {
5         Point p;
6         p = new Point();
7         Point p2;
8         Point p3;
9         Point p4;
10
11         p.abscisse = 1;
12         p.ordonnee = 2;
13         p.couleur = "rouge";
14         System.out.println(p);
15         p.mon_etat();
16         p.maj_etat(1, 2, "bleu");
17         System.out.println(p);
18         p.mon_etat();
19     }
20 }
```

The constructor Point() is undefined

5 quick fixes available:

- + Add arguments to match 'Point(double, double)'
- + Add arguments to match 'Point(double, double, String)'
- Change constructor 'Point(double, double)': Remove parameters 'double, double'
- Change constructor 'Point(double, double, String)': Remove parameters 'double, double'
- Create constructor 'Point()'

Press 'F'

Le constructeur avec paramètres

- si nous avons besoin d'un constructeur sans paramètres, il faut écrire son code.

```
Program.java  Point.java ✕
1 package com.setp1;
2
3 public class Point {
4     public double abscisse;
5     public double ordonnee;
6     public String couleur;
7
8     public Point(){
9         this.abscisse=0;
10        this.ordonnee=0;
11        this.couleur="noir";
12    }
13    public Point(double abscisse, double ordonnee){
14        this.abscisse=abscisse;
15        this.ordonnee=ordonnee;
16    }
17    public Point(double abscisse, double ordonnee,String couleur){
18        this.abscisse=abscisse;
19        this.ordonnee=ordonnee;
20        this.couleur=couleur;
21    }
22 }
```

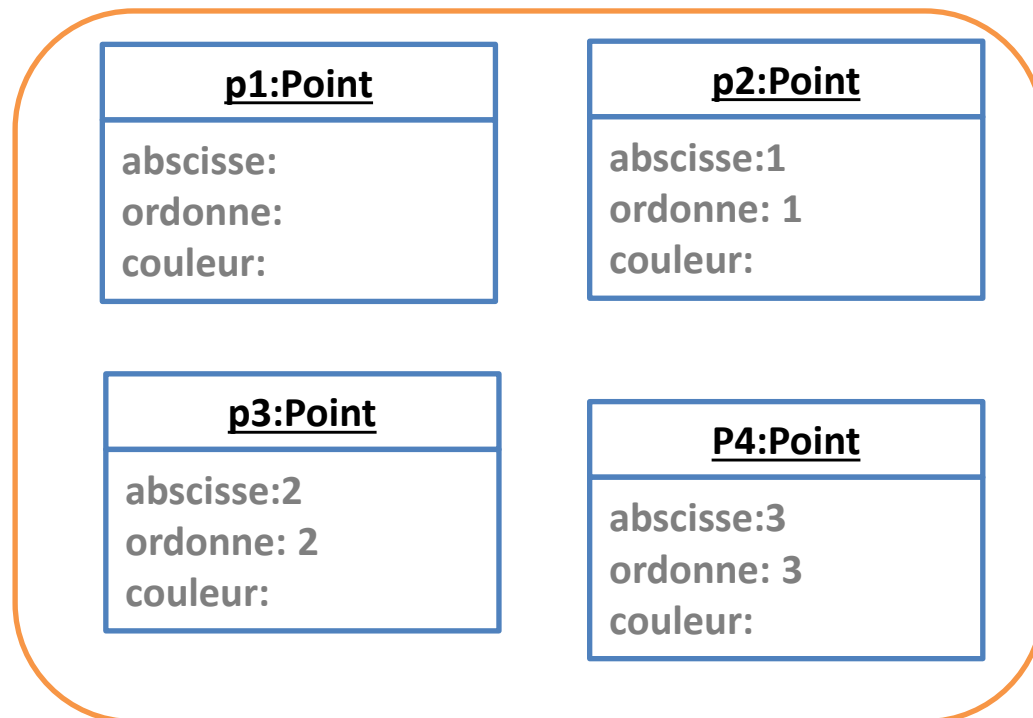
Les instructions du constructeur par défaut ne sont pas obligatoires. Nous pouvons mettre:

```
public class Point {
    public double abscisse;
    public double ordonnee;
    public String couleur;

    public Point(){
        |    }
    public Point(double abscisse, double ordonnee){
        this.abscisse=abscisse;
        this.ordonnee=ordonnee;
    }
}
```

Classe et Objet

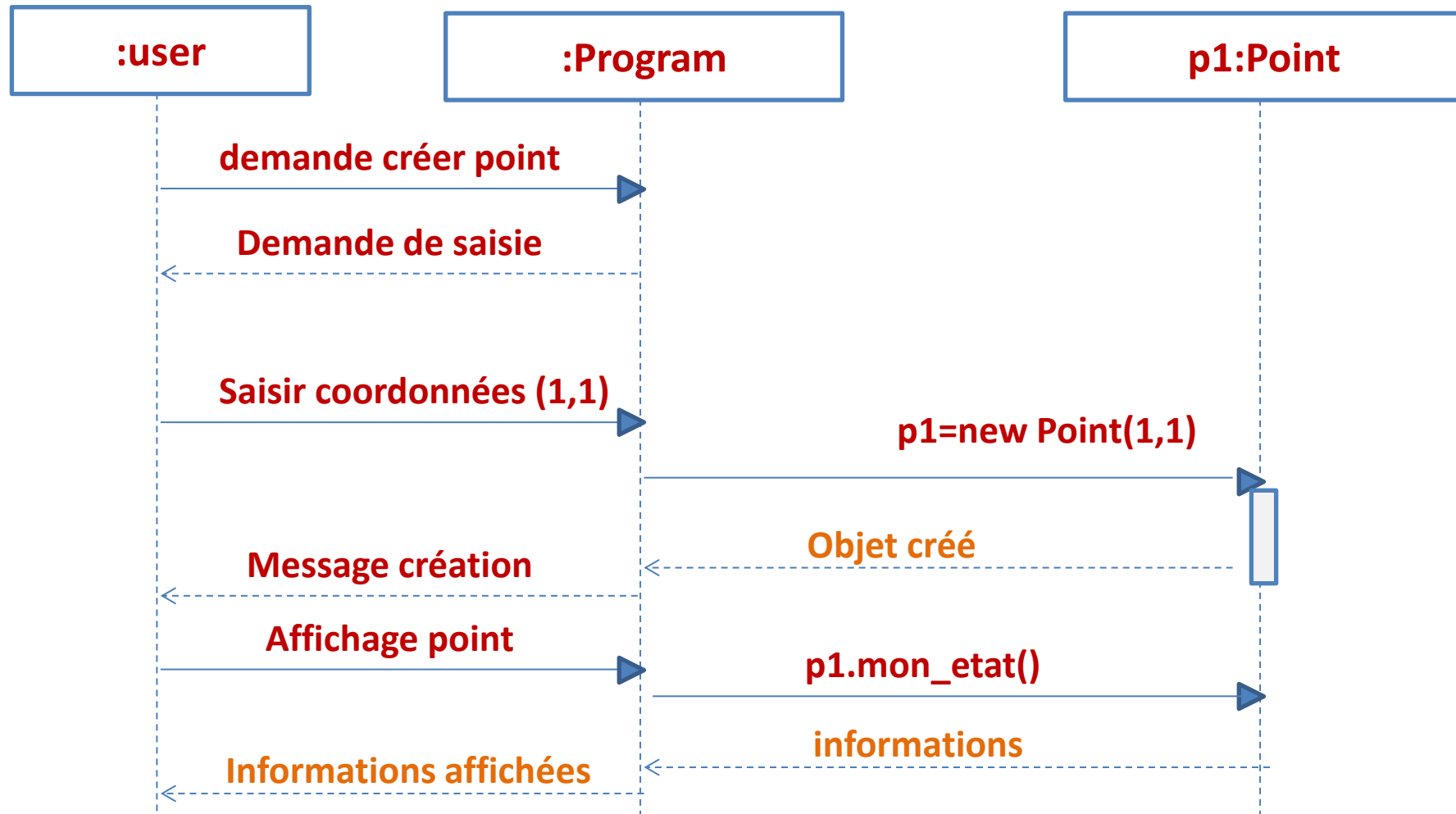
Diagramme d'objets



Classe et Objet

Diagramme de séquences

Un diagramme de séquences décrit les interactions entre les utilisateurs et les objets



Classe et Objet

Dans cette partie, nous allons traiter les concepts suivants:

- Classe
- Attributs
- Constructeur & Objet
- package
- Méthodes
- Objet courant
- Constructeur avec paramètres
- Membres statiques

NB: Durant cette partie, nous nous limitons à l'usage de la visibilité **public** des attributs

Les membres statiques

- Dans l'exemple de la classe Point, chaque objet Point possède ses propres variables abscisse, ordonnée et couleur. **Ces variables sont appelées variables d'instances.**
- Les objets d'une même classe peuvent **partager des mêmes variables** qui sont stockées au niveau de la classe. Ce genre de variables s'appellent les variables statiques ou variables de classes.
- Un attribut statique d'une classe est un attribut qui appartient à la classe et partagé par tous les objets de cette classe. Comme un attribut, une méthode peut être déclarée statique, ce qui signifie qu'elle appartient à la classe et partagée par toutes les instances de cette classe.
- Dans la notation UML, les membres statiques d'une classe sont soulignés.

Les membres statiques

Les membres statiques (attributs ou méthodes)

Un attribut statique

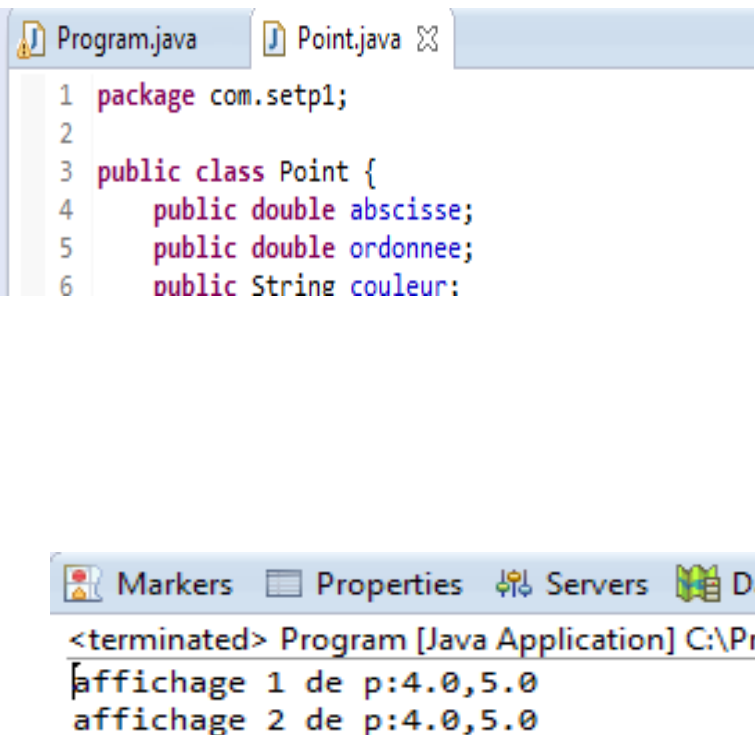
- Un attribut membre statique c'est une variable de la classe
- La valeur de la variable statique est indépendante de l'objet
 - Quelque soit le nombre d'objet créés, cette variable a une seule valeur
 - Cette variable peut avoir une valeur même si aucun objet n'existe

Les membres statiques

Les membres statiques (attributs ou méthodes)

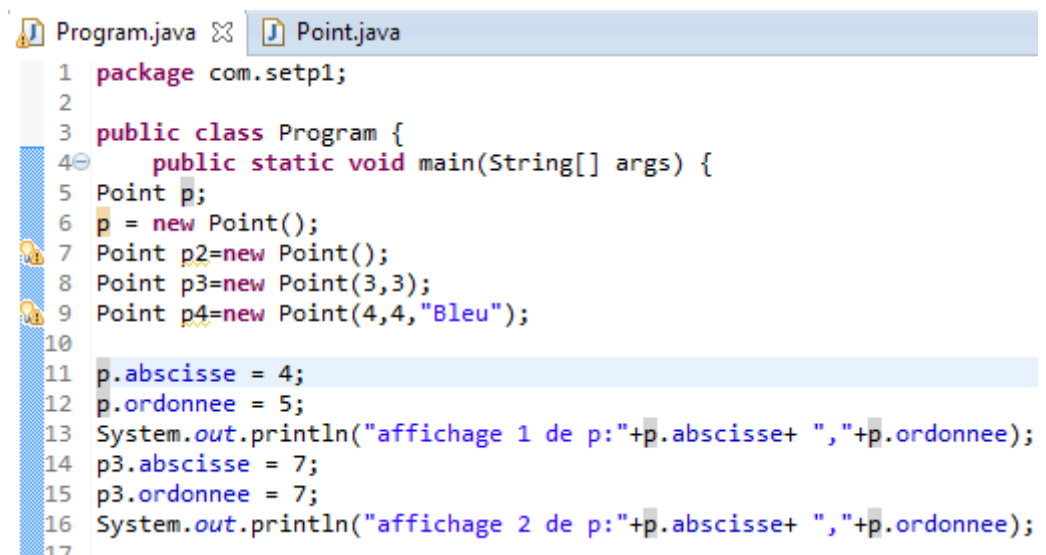
Un attribut non statique

- Un attribut membre non statique c'est une variable de l'objet



```
1 package com.setp1;
2
3 public class Point {
4     public double abscisse;
5     public double ordonnee;
6     public String couleur;
```

<terminated> Program [Java Application] C:\Pr
affichage 1 de p:4.0,5.0
affichage 2 de p:4.0,5.0



```
1 package com.setp1;
2
3 public class Program {
4     public static void main(String[] args) {
5         Point p;
6         p = new Point();
7         Point p2=new Point();
8         Point p3=new Point(3,3);
9         Point p4=new Point(4,4,"Bleu");
10
11         p.abscisse = 4;
12         p.ordonnee = 5;
13         System.out.println("affichage 1 de p:"+p.abscisse+ ","+p.ordonnee);
14         p3.abscisse = 7;
15         p3.ordonnee = 7;
16         System.out.println("affichage 2 de p:"+p.abscisse+ ","+p.ordonnee);
17     }
```

- La valeur de la variable statique est partagée entre tous les objets de la classe point

Les membres statiques

Les membres statiques (attributs ou méthodes)

Un attribut statique

- Un attribut membre statique c'est une variable de la classe

```
Program.java Point.java
1 package com.setp1;
2
3 public class Point {
4     public double abscisse;
5     public double ordonnee;
6     public String couleur;
7     public static int var_static;
8 }
```

```
Program.java Point.java
1 package com.setp1;
2
3 public class Program {
4     public static void main(String[] args) {
5         Point p;
6         p = new Point();
7         Point p2=new Point();
8         Point p3=new Point(3,3);
9         Point p4=new Point(4,4,"Bleu");
10
11
12         p.var_static=4;//p change la valeur de var_static
13         System.out.println("var_static depuis p2="+p2.var_static);
14         p3.var_static=6;
15         System.out.println("var_static depuis p2="+p2.var_static);
16
17     }
```

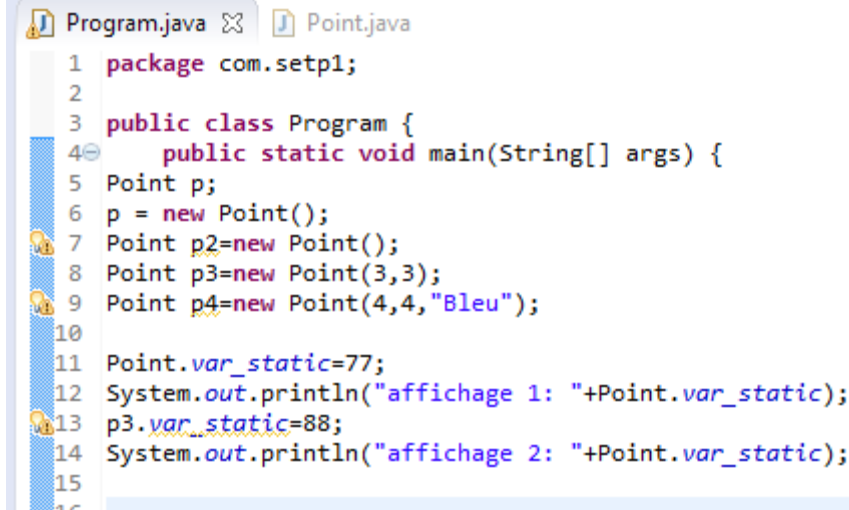
```
<terminated> Program [Java Application] C:\Program Files\Java\jdk1.7.0_
var_static depuis p2=4
var_static depuis p2=6
```

- La valeur de la variable statique est partagée entre tous les objets de la classe point

Les membres statiques

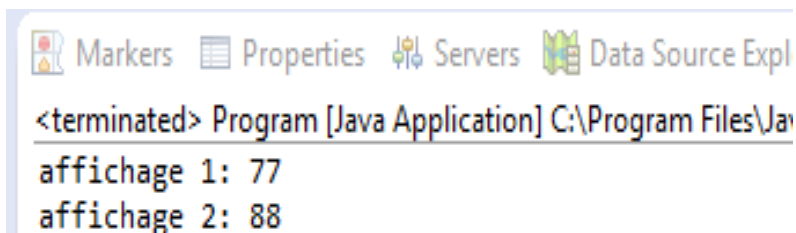
Les membres statiques (attributs ou méthodes)

Un attribut statique



```
1 package com.setp1;
2
3 public class Program {
4     public static void main(String[] args) {
5         Point p;
6         p = new Point();
7         Point p2=new Point();
8         Point p3=new Point(3,3);
9         Point p4=new Point(4,4,"Bleu");
10
11         Point.var_static=77;
12         System.out.println("affichage 1: "+Point.var_static);
13         p3.var_static=88;
14         System.out.println("affichage 2: "+Point.var_static);
15
16     }
```

- La variable « var_static » est accessible à travers le nom de la classe



```
<terminated> Program [Java Application] C:\Program Files\Ja
affichage 1: 77
affichage 2: 88
```

Les membres statiques

Les membres statiques (attributs ou méthodes)

Une méthode statique

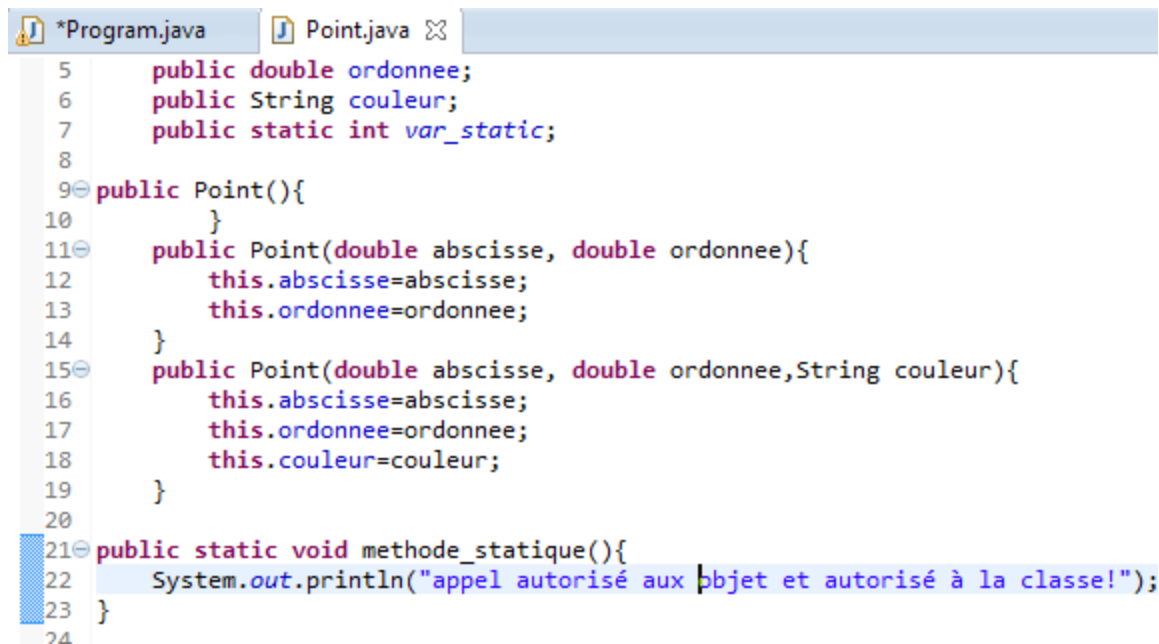
- Une méthode membre statique c'est une méthode de la classe
- Les tâches d'une méthode statique sont indépendantes des comportements des objets

Les membres statiques

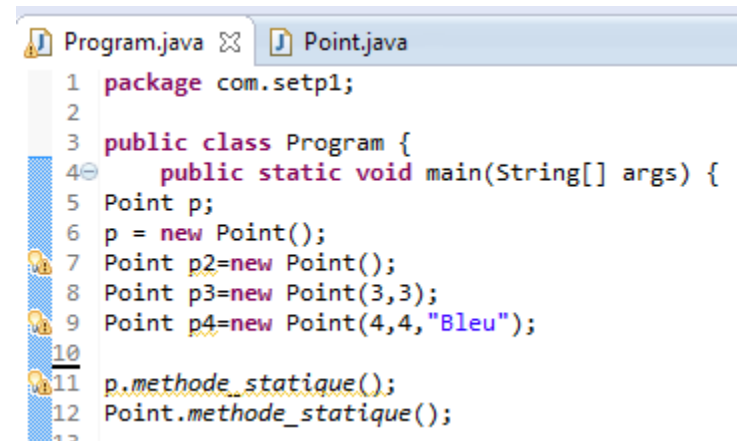
Les membres statiques (attributs ou méthodes)

Une méthode statique

- Une méthode membre statique c'est une méthode de la classe



```
*Program.java | Point.java ✕
5 public double ordonnee;
6 public String couleur;
7 public static int var_static;
8
9 public Point(){
10 }
11 public Point(double abscisse, double ordonnee){
12     this.abscisse=abscisse;
13     this.ordonnee=ordonnee;
14 }
15 public Point(double abscisse, double ordonnee,String couleur){
16     this.abscisse=abscisse;
17     this.ordonnee=ordonnee;
18     this.couleur=couleur;
19 }
20
21 public static void methode_statique(){
22     System.out.println("appel autorisé aux l'objet et autorisé à la classe!");
23 }
24
```



```
Program.java ✕ | Point.java
1 package com.setp1;
2
3 public class Program {
4     public static void main(String[] args) {
5         Point p;
6         p = new Point();
7         Point p2=new Point();
8         Point p3=new Point(3,3);
9         Point p4=new Point(4,4,"Bleu");
10
11         p.methode_statique();
12         Point.methode_statique();
13     }
14 }
```

- L'objet p peut accéder à la méthode « methode_statique() »

Les membres statiques

Les membres statiques (attributs ou méthodes)

- La valeur d'une variable statique est accessible par tous les objets de la classe à travers des méthodes non statiques
- Une méthode statique est accessible aux méthodes non statiques

- Exercice:

Des tests à faire pour vérifier ces propriétés

Les membres statiques

Les membres statiques (attributs ou méthodes)

- Les méthodes statiques ne peuvent accéder **directement** aux membres non statiques

```
20 public void acces_var_statique(){
21     var_static=4;
22 }
23
24 public void mon_etat()
25 {
26     methode_statique();
27
28     System.out.println ("mon état:");
29     System.out.println("(" + abscisse + ", " + ordonnee + ") " + couleur );
30 }
31 public static void methode_statique(){
32     System.out.println("appel autorisé aux objet et autorisé à la classe!");
33     abscisse=4;
34     mon_etat();
35 }
```

- La méthode `methode_statique()` ne peut pas accéder ni aux variables d'objets ni aux méthodes non statiques

Les membres statiques

Exercice

Écrit le code nécessaire pour comptabiliser le nombre de points créés

Classe-objet

Application N°1

Créer une application pour gérer les segments comme étant des formes géométriques

Un segment est défini par deux extrémités qui représentent chacun un point de l'espace géométrique et par une couleur.

- Créer le diagramme de cas d'utilisation, le diagramme de classe, le diagramme d'objets et le diagramme de séquence de votre projet.
- Dans cette application, il faut faire appel aux différentes techniques introduites à l'étape 1 et 2. pour chaque bout de code écrit, préciser le concept introduit.

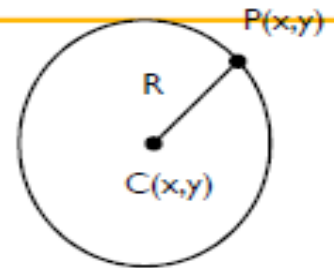
L'application doit permettre les opérations suivantes:

- Créer un segment (**Constructeurs**)
- Afficher les informations d'un segment (**toString()**)
- Calculer la longueur d'un segment (**getLongueur()**)
- Vérifier si un point appartient au segment (**Appartient (Point p)**)
- Comptabiliser le nombre de segments créés
- Créer un scénario d'exécution à l'aide d'une classe contenant le main()

Classe-objet

Application N°2

- Une cercle est défini par :
 - Un point qui représente son centre : centre(x,y) et un rayon.
 - On peut créer un cercle de deux manières :
 - Soit en précisant son centre et un point du cercle.
 - Soit en précisant son centre et son rayon
 - Les opérations que l'on souhaite exécuter sur un cercle sont :
 - `getPerimetre()` : retourne le périmètre du cercle
 - `getSurface()` : retourne la surface du cercle.
 - `appartient(Point p)` : retourne si le point p appartient ou non à l'intérieur du cercle.
 - `toString()` : retourne une chaîne de caractères de type `CERCLE(x,y,R)`
1. Etablir le diagramme de classes
 2. Créer la classe Point définie par:
 - Les attributs x et y de type int
 - Un constructeur qui initialise les valeurs de x et y.
 - Une méthode `toString()`.
 3. Créer la classe Cercle
 4. Créer une application qui permet de :
 - a. Créer un cercle défini par le centre c(100,100) et un point p(200,200)
 - b. Créer un cercle défini par le centre c(130,100) et de rayon r=40
 - c. Afficher le périmètre et le rayon des deux cercles.
 - d. Afficher si le point p(120,100) appartient à l'intersection des deux cercles ou non.



Conclusion

Dans cette partie, nous avons traité les concepts suivants:

- Classe
- Attributs
- Constructeur & Objet
- package
- Méthodes
- Objet courant
- Constructeur avec paramètres
- Membres statiques

NB: Durant cette partie, nous nous sommes limités à l'usage de la visibilité **public** des attributs

Programmation Orientée Objets avec **JAVA**

Fin du chapitre 2