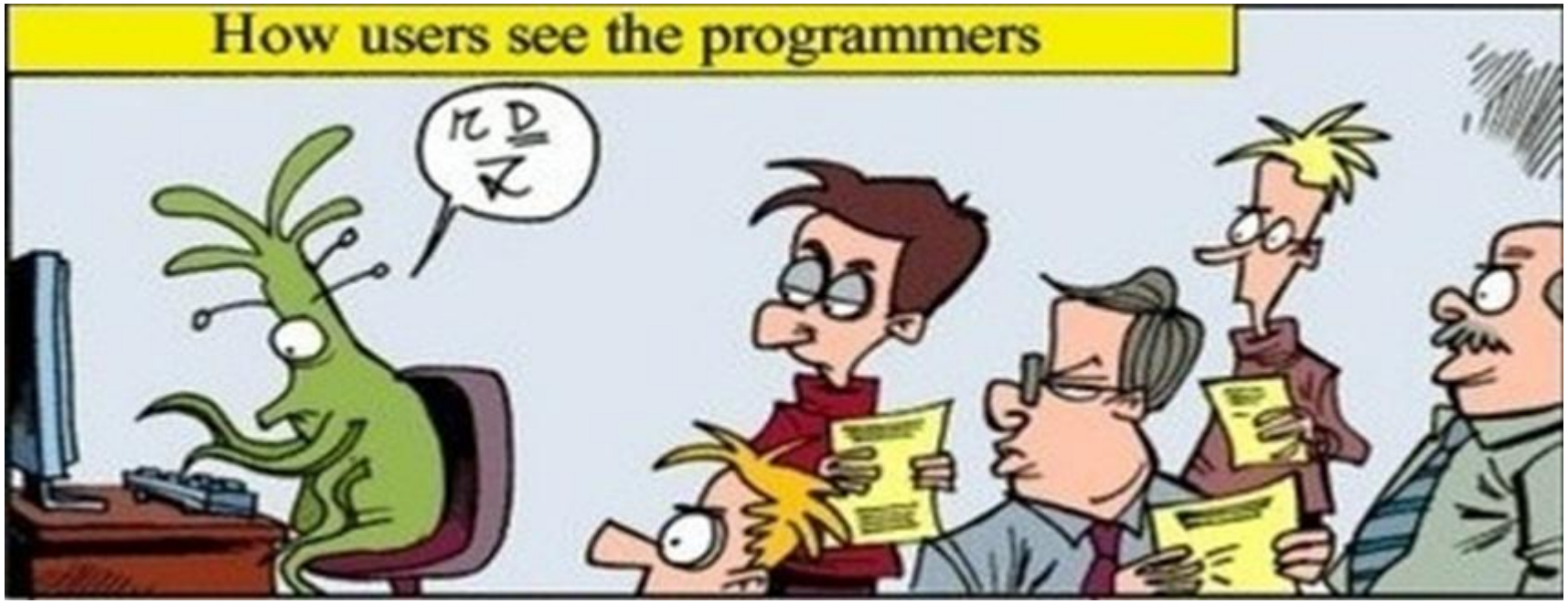
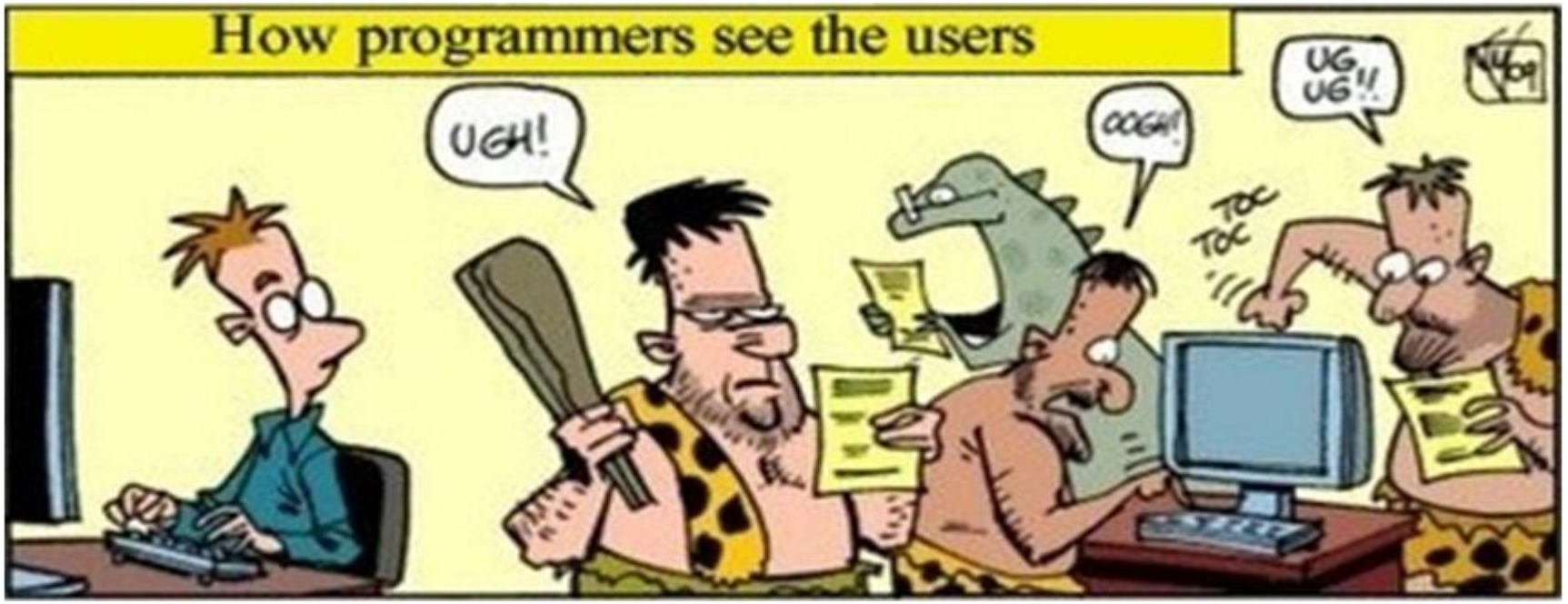
Cours de POO avec Java (Partie 2)

Module GL et POO
2LMD,
2014
Kahloul Laid

Avant de commencer

Questions?


How users see the programmers


How programmers see the users
UGH!
OOGH!
UG!! UG!!
TOC TOC

Mes Questions:

- JVM?
- JDK?
- Quelle contrainte sur le nom du fichier en java?
- Y’a-t-il un programme principal en java?
- Comment exécuter un programme en java?

Objectif

Avancer dans la programmation OO avec Java : instanciation, visibilité entre classe, héritage

Plan

- **Constructeur et destructeur**
- **Instanciation** : création d'objet par le programmeur;
- **Visibilité entre classe**: différents types de visibilité;
- **Héritage entre classe**: c'est quoi?, méthode virtuelle, revoir la visibilité**.**

Le constructeur (1)

- Une méthode qui doit exister dans **toute classe**;
- Elle est exécutée pour **créer et initialiser** les instances (les objets);
- Elle porte **le même nom de la classe**;
- Si elle n'est pas définie par le programmeur, elle est **ajouté implicitement** par le système
- Pour appeler le constructeur :

  new nom_constructor(args);

Le constructeur (2):

exemple d'un constructeur défini par le programmeur

Exemple:

```
class cercle{
 // liste des attributs;
        int x; int y; int r;
 // liste des méthodes;
     // un constructeur
      cercle(int x0, int y0, int r0){
      x= x0;
      y= y0;
      r= r0;
      }
 }
```

Un constructeur

Le destructeur (1)

- Une méthode qui **doit exister dans toute classe**;
- Elle est exécutée pour **détruire** l'objet;
- Exemple: en C++ elle porte le même **nom de la classe précédé** par le **symbole ~**; Mais en Java: **pas de destructeur explicite**. Le système s'occupe de la destruction (**ramasse miettes**).
- **le système (JVM)**: réalise la destruction des instance à la fin de leurs usages
- Le programmeur peut utiliser la méthode **finalize()** pour seulement **voir quand son objet est détruit** par le système (**chose non garantie**).

Le destructeur (2):

exemple d'un destructeur défini par le programmeur

Exemple:

```
class cercle{
 // liste des attributs;
     int x; int y; int r;
 // liste des méthodes;
      // un destructeur
   public void finalize(){
   // l'objet est détruit
   }
 }
```

Constructeur et destructeur :

Exemple en java

Voir les fichiers associés à ce cours:

- **cercle2.java**: une classe avec un constructeur, appelé plusieurs fois
- **cercle3.java**: une classe avec un constructeur et un destructeur
- Essayez de tester ces deux programmes avec **javac** et **java**

Instanciation d'objet (1)

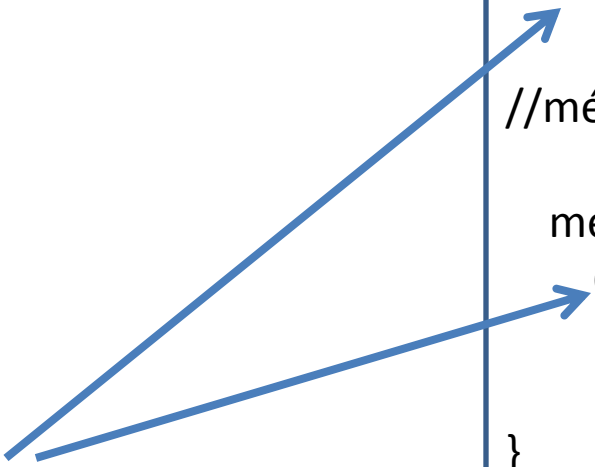
- Il est possible de créer **autant d'instances** qu'on veut d'une classe;

- Un **objet o1 de C1** peut être créé dans la définition de la classe **C1** ou dans une autre classe **C2**.

Instanciation d'objet (2)

```
class C1{

//attributs
//méthodes

}
```

```
class C2{

//attributs

    C1 o1=new  C1();

//méthodes

   methode (){
     C1 o1=new  C1();
          }

}
```



Instanciation en deux lieux différents

Instanciation d'objet (3)

```
class cercle{
        // liste des attributs;
    int x; int y; int r;
        // liste des méthodes;
        // un constructeur
    cercle(int x0, int y0, int r0){
            x= x0; y= y0; r= r0;
            }
        }
```

```
class rectangle{

        // liste des attributs;
            int x0; int y0; int l1; int l2;

            cercle  c1=new cercle(1,3,5);

        // liste des méthodes;

            void method1(){
            cercle  c2=new cercle(3,9,10);

            }
}
```

Une instance

Une autre instance

Instanciation d'objet (4): exemple

- Voir le fichier **rectangle.java**
- **Attention:** le nom du fichier porte le nom de la classe qui contient la méthode **main**

Visibilité entre classe
Les modificateurs: **public** vs **private**

- Les attributs et les méthodes d'une classe peuvent être visibles dans toutes les autres classes: dans ce cas il sont précédés par le **modificateur public**

- Les attributs et les méthodes d'une classe peuvent être invisible dans toutes les autres classes: dans ce cas il sont précédés par le **modificateur private**

Visibilité entre classe: exemple

```
class cercle{
        // liste des méthodes;
    public int x;
    private int y;
    int r;
        // liste des méthodes;
       // un constructeur
      cercle(int x0, int y0, int r0){
                    x= x0; y= y0; r= r0;
                    }
        }
```

```
class rectangle{

        // liste des méthodes;
            int x0; int y0; int l1; int l2;

    // liste des méthodes;

            void method1(){
            cercle  c2=new cercle(3,9,10);
            c2.x=5;
            c2.y=6;
            }
}
```

Accès possible car le **x est public**

Accès impossible car **y est privé**

Visibilité entre classe: exemple

1) Voir le fichier rectangle2.java;

2) Compiler et voir le genre d'erreurs produites;

3) Expliquer les causes de ces erreurs

Héritage (1)

- Pour la définition: revoir **le premier cours de POO**;
- Une classe A peut hériter une classe B: **A va disposer de tous les attributs et les méthodes de B**;
- En **java**, on dispose **uniquement de l'héritage simple**: une classe ne peut pas hériter de plusieurs classes;
- Dans le constructeur de la classe fille, on doit faire appel explicitement au constructeur de classe mère, en utilisant le mot clé **super**;

Héritage (2): mot clé **extends**

```
class cercle{
        // liste des méthodes;
    int x;
    int y;
    int r;
      // liste des méthodes;
      // un constructeur
     cercle(int x0, int y0, int r0){
       x= x0; y= y0; r= r0;
                    }
      }
```

```
class cercle_colore extends cercle{

        // liste des méthodes;
              int c;

        // liste des méthodes;

    cercle_colore(int x0, int y0, int r0 , int c0 ){
              super(x0, y0, r0);
              c= c0;
              }
}
```

Mot clé de l'héritage

En plus de **x, y, r** l'attribut **c** est ajouté à cette classe fille

Appel obligatoire au **constructeur de la superclasse**

Héritage (3): méthodes virtuelles

- Une méthode virtuelle est une méthode **définie** dans la **classe mère A**, ensuite **redéfinie** dans la **classe fille B**;

- En **Java**, **toutes les méthodes** d'une classe sont **virtuelles**.

Héritage (4):
exemple: **méthode virtuelle**

```
class cercle{
        // liste des méthodes;
      int x;  int y;  int r;
        // liste des méthodes;
        // un constructeur

   cercle(int x0, int y0, int r0){
                        x= x0; y= y0; r= r0;
                        }

   void changer_attribut(int x1, int y1, int r1){
          x= x1; y= y1; r= r1;
          }

}
```

```
class cercle_colore extends cercle{
          // liste des méthodes;
                int c;
        // liste des méthodes;
         cercle_colore
           (int x0, int y0, int r0 , int c0 )
           {super(x0, y0 ,r0); c= c0;  }

void changer_attribut
          (int x1, int y1, int r1 , int c1)
       { x= x1; y= y1; r= r1 ; c= c1; }
}
```

Une méthode virtuelle

méthode redéfinie dans la classe fille

Héritage (5):
exemple: cercle_colore.java

- Voir le fichier **cercle_colore.java**;

- Exécuter les deux méthodes sur deux instances différentes de deux classes **cercle**, et **cercle_colore**;

- Ajouter une méthode **affichage** pour afficher les attributs des deux objets;

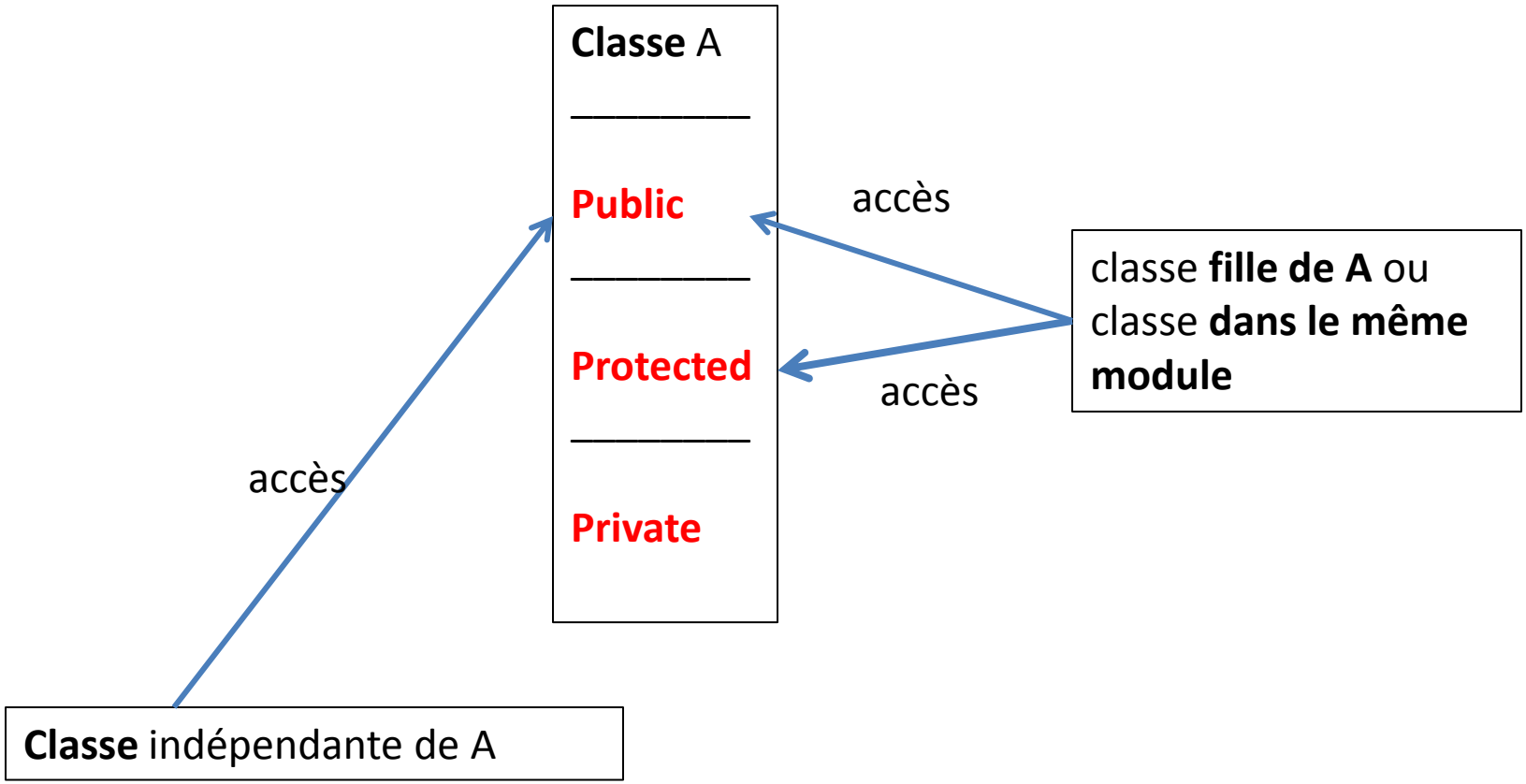
Héritage (6) : Revoyant la visibilité

- La visibilité des attribut prend une autre dimension;

- Un **autre modificateur** est ajouté: **protected** (qui peut précéder les noms des attributs ou des méthodes);

- Un attribut **protected** est **accessible** dans la **classe A** et dans ses **classes filles**, et les **calsses de son module** (concept à revoir dans les prochains cours).

Héritage (7) : héritage et visibilité



Héritage (8) : exemple

- Voir le fichier rectangle3.java;
- Compiler le fichier;
- Essayer de comprendre les erreurs;
- Corriger les erreurs.