La communication objet

1. Les données static

```
public class Cercle
{ public int x, y, r;
  public static int nombre;

  public void creer()
  { .println(" Position en x : ");
    x=Lire.i();
    .println(" Position en y : ");
    y=Lire.i();
    .println(" Rayon : ");
    r=Lire.i();
    nombre ++;
  }
}
```

```
public class CompterDesCercle
{  public static void main (String [] arg)
   { Cercle A = new Cercle();
     A.creer();
     .println(" Nb. de c : " + Cercle.nombre);
     Cercle B = new Cercle();
     B.creer();
     .println(" Nb. de c : " + Cercle.nombre);
   }
}
```

Grâce au mot-clé **static**, la variable de classe **nombre** a un espace mémoire commun, accessible à tous les objets créés. Pour faire appel: **nombre** ou **Cercle.nombre**

2. Le passage des paramètres par référence

Ce procédé est utilisé lorqu'on passe en paramètre d'une méthode, non plus une simple variable (de type `int, char, double`), mais un objet. Dans cette sitution, l'objet étant défini par son adresse (référence), la valeur passée en paramètre n'est plus la valeur réelle de la variable mais l'adresse de l'objet.

Grâce à cela, les modifications apportées sur l'objet passé en paramètre et réalisées à l'intérieur de la méthode sont visibles en dehors même de la méthode.

```
public class Cercle
{ public int x, y, r;
  public static int nombre;

  public void creer()
  { .......
  }
  public void echange(Cercle autre)
  { int tmp;
    tmp=x;
    x=autre.x;
    autre.x=tmp;
    tmp=y;
    y=autre.y;
    autre.y=tmp;
  }
}
```

```
public class EchangerCercles
{  public static void main (String [] arg)
   { Cercle A = new Cercle();;
     A.creer();
     .println(" Le cercle A : ");
     A.afficher();
     Cercle B = new Cercle();;
     B.creer();
     .println(" Le cercle B : ");
     B.afficher();

     B.echange(A);

     .println(" Le cercle A : ");
     A.afficher();
     .println(" Le cercle B : ");
     B.afficher();
   }
}
```

Les objets contrôlent leur fonctionnement

1. protection des données

• Protection **public** Les membres (données et méthodes) d'une classe déclarée **public** sont accessibles pour tous les objets de l'application. Les données peuvent être modifiées par une méthode de la classe, une autre classe ou depuis la fonction main().

• Protection **private** Les membres d'une classe déclarés **private** ne sont accessibles que pour les méthodes de la même classe. Les données ne peuvent être initialisées ou modifiées que par l'intermédiaire d'une méthode de la classe. Les données ou méthodes déclarées **private** ne peuvent être appelées par la fonction main().

• Protection **protected** Comme le membres privés, les membres déclarés **protected** ne sont accessibles que pour les méthodes de la même classe et aussi par les membres d'une sous-classe.

```
public class CerclePrive
{ private int x, y, r;
  public void afficher()
  {  ....}
  public void perimetre()
   {  ....}
  public void deplacer(int nx, int ny)
   {  ....}
  public void agrandir(int nr)
   {  ....}
}


public class FaireDesCerclesPrives
{  public static void main (String [] arg)
   { CerclePrive A = new CerclePrive();
     A.afficher();
     .println(" Entrez le rayon : ");
     A.r=Lire.i();
     .println(" Le rayon est : "+A.r)
   }
}
```

2. Les méthodes d'acces aux données

• Accès en consultation La méthodes fournit la valeur de la donnée mais ne peut la modifier

• Accès en modification La méthode modifie la valeur de la donnée. Cette modification est réalisée par la méthode

```
public class CercleControle
{ private int x, y, r;
  public void creer()
  { .println(" Position en x: ");
    x=Lire.i();
    .println(" Position en y: ");
    y=Lire.i();
    do
    { .println(" Rayon: ");
      r=Lire.i();
    } while ( r < 0 || r > 600 );
   }
  public void afficher()
  {  ....}
  public void agrandir(int nr)
  { if (r+nr < 0) r=0;
    else if (r+nr > 600) r=600;
    else r = r +nr;
  }
}
```

```
public class FaireDesCerclesControles
{  public static void main (String [] arg)
   { CerclePrive A = new CercleControle();
     A.creer();
     A.afficher();
     .println(" Valeur d'agrandissement: ");
     int plus = Lire.i();
     A.agrandir(plus);
     .println(" Apres l'agrandissement: ");
     A.afficher();
   }
}
```

La notion de constante

En programmation orientée objet les variables d'une classe ne sont que très rarement déclarées en **public**. Pour des raisons de sécurité, tout objet se doit de contrôler les transformations opérées sur ses variables. C'est pourquoi les données d'une classe sont le plus souvent déclarées en mode **private**. Les données constantes sont declarées **public** mais l'application ne peut pas modifier leur contenu.

```
public final int TailleEcran = 600;

public final static int TailleEcran = 600;
```

```
public void creer()
  { .println(" Position en x: ");
    x=Lire.i();
    .println(" Position en y: ");
    y=Lire.i();
    do
    { .println(" Rayon: ");
      r=Lire.i();
    } while ( r < 0 || r > TailleEcran );
   }
  public void agrandir(int nr)
  { if (r+nr < 0) r=0;
    else if (r+nr > TailleEcran)
    r=TailleEcran;
    else r = r +nr;
  }
```

Méthodes invisibles

```
private int rayonVerifie()
{ int temp;
  do
 { .println(" Rayon: ");
    temp=Lire.i();
 } while ( temp < 0 || temp > TailleEcran );
   return temp;
}

private int rayonVerifie(int temp)
{ if ( temp < 0 )
  return 0;
  else if ( temp > TailleEcran)
  return TailleEcran;
  else
  return temp;
}
```

Les constructeurs

```
Cercle C =  new Cercle();

public Cercle()
{ .println(" Position en x: ");
  x = Lire.i();
  .println(" Position en y: ");
  y = Lire.i();
  r = rayonVerifie()
}
```

La surcharge de méthodes

```
public Cercle(int cx, int cy)
{ x = cx;
  y = cy;
}

public Cercle(int cx, int cy, int rayon)
{ x = cx;
  y = cy;
  r = rayon;
}
```

Le mot-clé this

```
public Cercle(int cx, int cy, int rayon)
{ this(cx, cy);
  r = rayon;
}
```

• **this** repésente l'appel a la méthode Cercle

• **this** doit être placé comme première instruction du constructeur qui l'utilise

L'héritage

Avec l'héritage, les méthodes définies pour un ensemble de données sont réutilisables pour des variantes de cet ensemble. Par exemple, si nous supposons qu'une classe **Forme** définisse un ensemble de comportement propres à toute forme géometrique, alors :

• Ces comportements peuvent être utilisés par la classe **Cercle**, qui est une forme géométrique particulière. Cette réutilisation est effectuée sans à modifier les instruction de la classe **Forme**

• Il est possible d'ajouter d'autres comportements spécifiques des objets **Cercle**. Ces nouveaux comportements sont valides uniquement pour la classe **Cercle** et non plus pour la classe **Forme**.

```
class B extends A
{ //données et méthodes de la classe B
```

• B est une **sous-classe** de A ou est une **classe dérivée** de A

• A est une **super-classe** ou une **classe de base**

```
public class Forme
{ protected int x, y;
  private couleur;

  public Forme()
  { .println("position en x ");
    x = Lire.i();
    .println("position en y ");
    y = Lire.i();
    .println("couleur de la forme ");
    couleur = Lire.i();
  }

  public void afficher()
  { .println("position "+x+","+y);
    .println("couleur"+couleur);
  }

  public void deplacer(int nx, int ny)
  { x=nx;
    y=ny;
  }
}
```

```
public class Cercle extends Forme
{ private int r;

  public Circle()
  { r = rayonVerifie();
  }

  private int rayonVerifie()
  { ...}

  public void afficher()
  { super.afficher();
    .println("rayon "+r);
  }
}
```

Le mot-clé super

```
public Cercle (int xx, int yy)
{ super(xx, yy);
  .printle("Rayon ?");
  r=rayonVerifie();
}
```

• Le terme `super` est obligatoirement la première instruction du constructeur de la classe derivée

• La liste des paramètres (deux `int`) permet de préciser au compilateur quel est le constructeur utilisé en cas de surcharge de constructeur.

La protection des données héritées

```
public Cercle (int xx, int yy)
{ super(xx, yy);
  couleur = 20;
  r=10;
}
```

- Si `couleur` a été declaré `private` **erreur** !
- Si `couleur` a été declaré `protected` OK !

Le polymorphisme

La notion de polymorphisme découle directement de l'héritage. Par polymorphisme, il faut comprendre qu'une méthode peut se comporter différemment suivant l'objet sur lequel elle est appliquée.

Lorsqu'une même méthode est définie à la fois dans la classe mère et dans la classe fille, l'exécution de la méthode choisie est réalisée en fonction de l'objets associé à l'appel et non plus suivant le nombre et le type des paramètres (voir surcharge)

Le polymorphisme

```
public classe FormerDesCercles
{ public static void main (String [] arg)
  { Cercle A = new Cercle(5,5);
    A.fficher();
    Forme F = new Forme(10,10, 3);
    F.fficher();
  }
}
```