Université de Bretagne Occidentale

Java

EN QUELQUES SLIDES

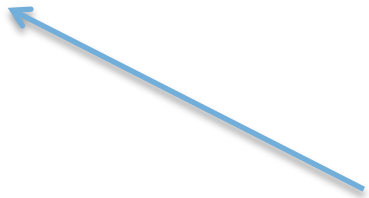
Ahcène Bounceur

ECRITURE : NORMALISATION

- Classe
  - Le nom d'une classe doit commencer par une lettre majuscule
    - Point, Rectangle, Personne
  - S'il est composé de plusieurs mots, chaque mot doit commencer par une majuscule
    - RectangleArrondi, ToucheEnPlastique, TableauDeBord
  - Toutes les lettres d'une constante doivent être en majuscule (on peut utiliser le caractère _)
    - NOIR, VERT, HAUT, BAS, GAUCHE, AC, ZERO, ROUGE_VIF
  - Le nom d'une variable ou d'un objet doit commencer par une lettre minuscule
    - rectangleArrondu, toucheEnPlastique, tableauDeBord

ECRITURE : NORMALISATION

- Exemple d'une déclaration d'une classe :
  - Point point ;
  - Point point1;
  - Point point_1;
  - Point pointRouge;
  - Point point_rouge;
  - TableauDeBord tableauDeBord ;
  - TableauDeBord taleau_de_bord ;
  - final TableauDeBord TABLEAU_ROUGE;



constante

UNE CLASSE

- Classe

```
public class Point {

}
```

UNE CLASSE

- Attributs

```
public class Point {

        private int x ;
        private int y ;
}
```

```
public class A {
   public void m() {
      Point p = new Point();
      p.x = 4;
   }
}
```

Encapsulation

Un autre objet ne pourra ni accéder ni modifier par accès direct les valeurs de x et y
Il doit passer par les getters et les setters

UNE CLASSE

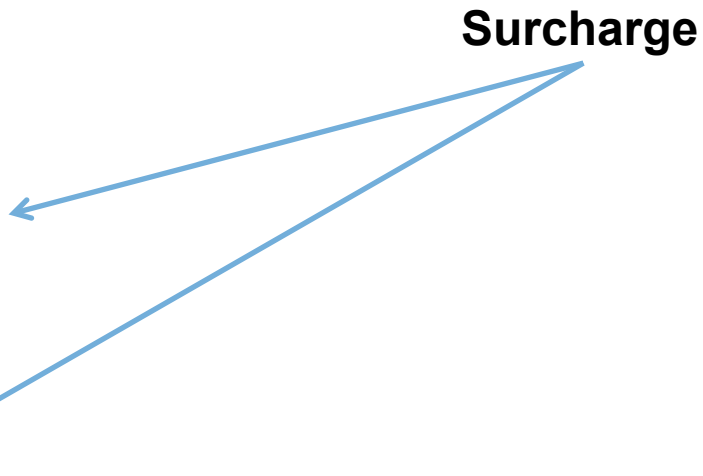
- Constructeur vide

```
public class Point {

        private int x ;
        private int y ;
        public Point() {
                x = y = 0;
        }
}
```

UNE CLASSE

- Constructeurs (surcharge)

```
public class Point {

        private int x ;
        private int y ;
        public Point() {
                x = y = 0;
        }

        public Point(int x, int y) {
                this.x = x;
                this.y = y;
        }
}
```



UNE CLASSE

- Getters/Setters (accesseurs)

```
public class Point {
        ...
        public void setX(int x) {
                this.x = x;
        }
        public void setY(int y) {
                this.y = y;
        }
}
```

UNE CLASSE

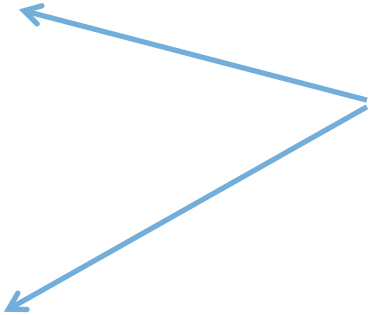
- Getters/Setters (accesseurs)

```
public class Point {
        ...
        public int getX() {
                return x;
        }
        public int getY() {
                return y;
        }
}
```

UNE CLASSE

- Méthodes (**Surcharger** une méthode)

```
public class Point {
        ...
        public void deplacer(int x, int y) {
                this.x = x;
                this.y = y;
        }

        public void deplacer(int d) {
                x = x + d ;
                y = y + d ;
        }
}
```

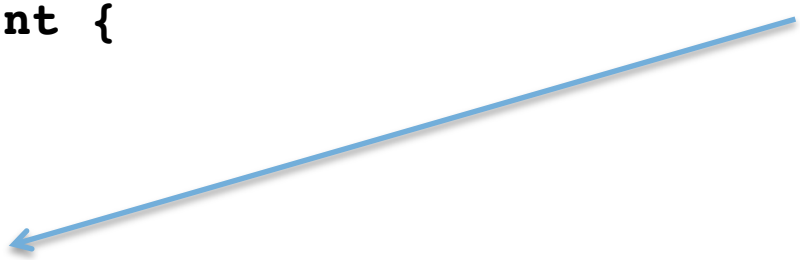


UNE CLASSE

- Méthodes (**redéfinition** de toString())

```
public class Point {
        ...

        @Override
        public String toString() {
                return "("+x+", "+y+")";
        }

}
```

annotation



*Cette méthode est déjà définie dans la classe **Object***
*Toutes les classes héritent par défaut de la classe **Object***

LES OBJETS

- Un objet se crée par **instanciation** :

```
Point p = new Point();
```

p (x=0, y=0)

→ Fait appel au constructeur vide.

```
Point p = new Point(2, 3);
```

p (x=2, y=3)

→ Fait appel au deuxième constructeur.

Point est une classe

p est un objet

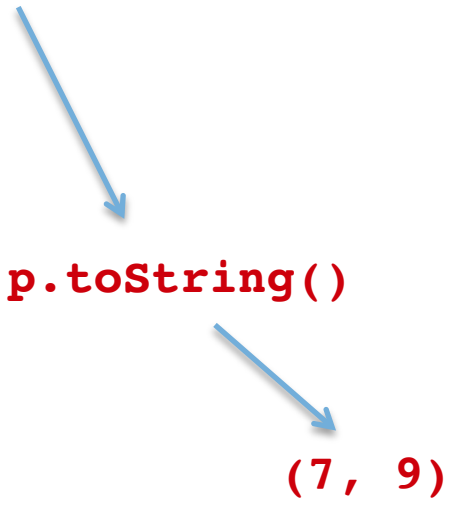
La Classe Principale : main

```
public class ClassePrincipale {

        public static void main(String[] args) {

        }

}
```

Point d'entrée

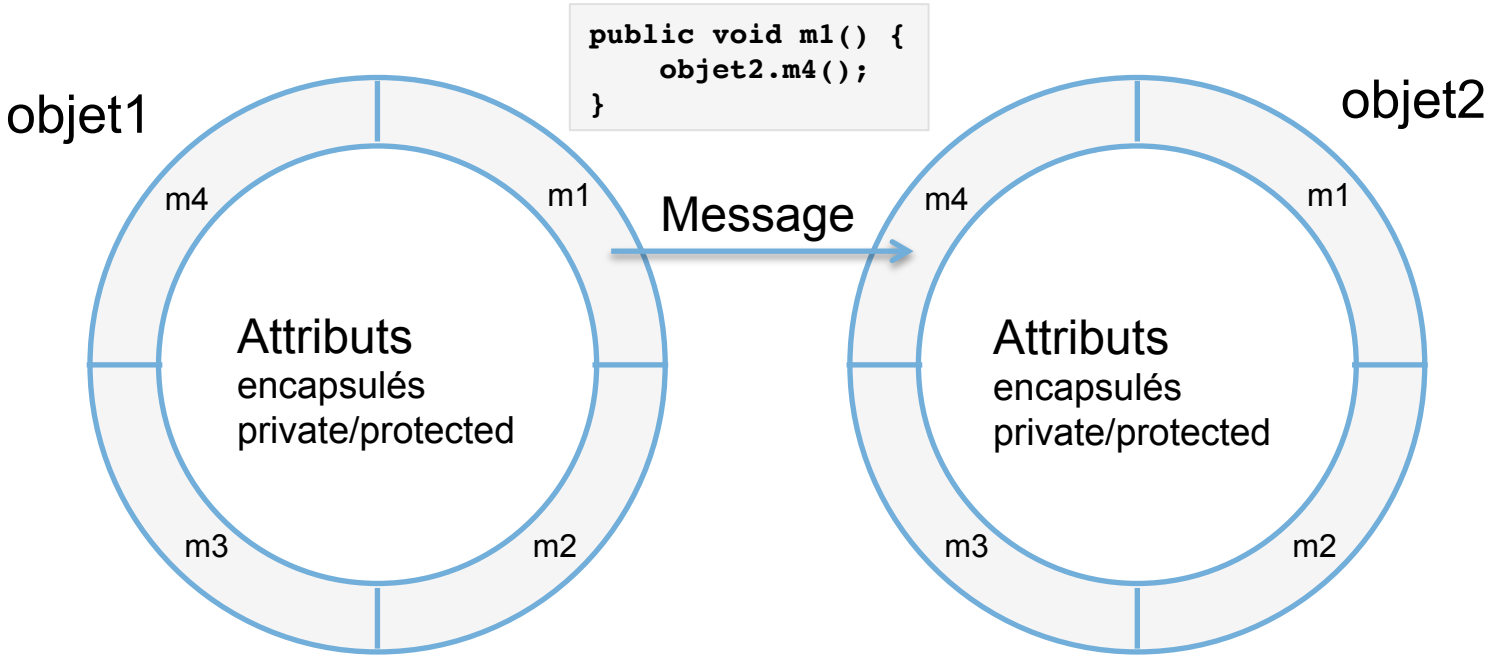
LA CLASSE PRINCIPALE : MAIN

```
public class ClassePrincipale {

        public static void main(String[] args) {
                Point p = new Point(3,5);
                p.deplacer(4);
                System.out.println(p);
        }

}
```



LES OBJETS

- Communication et messages :
  - Un objet ne doit pas modifier la valeur d'un attribut d'un autre objet (principe d'encapsulation)
  - Il doit utiliser les messages
  - Un message est un appel à une méthode
  - Les objets communiquent par les méthodes



CLASSE ABSTRAITE

- Classe Figure

*Un figure se **dessine**, mais on ne sait pas la dessiner encore.*


```
public abstract class Figure {

        protected Color couleur ;
        protected Point position;

        // Constructeurs

        public abstract void dessiner(Graphics g);

        // Getters/Setters
}
```

```
Figure()
Figure(x, y)
Color getCouleur()
Point getPosition()
Void setCouleur(couleur)
Void setPosition(position)
```

CLASSE ABSTRAITE

- Attribut de classe

```
public abstract class Figure {
        public static int zoom ;
        protected Color couleur ;
        protected Point position;

        // Constructeurs

        public abstract void dessiner(Graphics g);

        // Getters/Setters
}
```

zoom est un attribut qui appartient à la classe et non pas aux objets qui seront instanciés

```
Figure()
Figure(x, y)
Color getCouleur()
Point getPosition()
Void setCouleur(couleur)
Void setPosition(position)
```

CLASSE ABSTRAITE

- Méthode de classe

```
public abstract class Figure {
        public static int zoom ;
        protected Color couleur ;
        protected Point position;
        // Constructeurs
        public static void setZoom(int zoom) {
                Figure.zoom = zoom;
        }
        public abstract void dessiner(Graphics g);
        // Getters/Setters
}
```

zoom est un attribut qui appartient à la classe et non pas aux objets qui seront instanciés

```
Figure()
Figure(x, y)
Color getCouleur()
Point getPosition()
Void setCouleur(couleur)
Void setPosition(position)
```

CLASSE ABSTRAITE

- Méthode de classe

```
public abstract class Figure {
        public static int zoom ;
        protected Color couleur ;
        protected Point position;
        // Constructeurs
        public static int getZoom() {
                return zoom;
        }
        public abstract void dessiner(Graphics g);
        // Getters/Setters
}
```

zoom est un attribut qui appartient à la classe et non pas aux objets qui seront instanciés

```
Figure()
Figure(x, y)
Color getCouleur()
Point getPosition()
Void setCouleur(couleur)
Void setPosition(position)
```

CLASSE ABSTRAITE

Figure

- Méthodes abstraites

```
public abstract class Figure {
        public static int zoom ;
        protected Color couleur ;
        protected Point position;

        public abstract void dessiner(Graphics g);
        public abstract double getSurface();

}
```

zoom est un attribut qui appartient à la classe et non pas aux objets qui seront instanciés

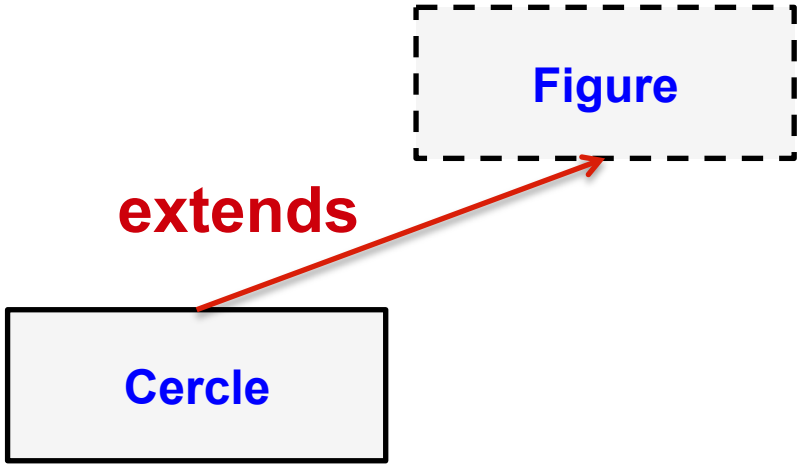
CLASSE ABSTRAITE

- **toString()**

```
public abstract class Figure {
        public static zoom ;
        protected Color couleur ;
        protected Point position;

        public abstract void dessiner(Graphics g);
        public abstract double getSurface();
        @Override
        public String toString() {
                return getClass().getName()+"("
                +position.getX()+", "
                +position.getY()+") ["+getSurface()+")";
        }
}
```

L'HÉRITAGE

- L'héritage : Classe Cercle

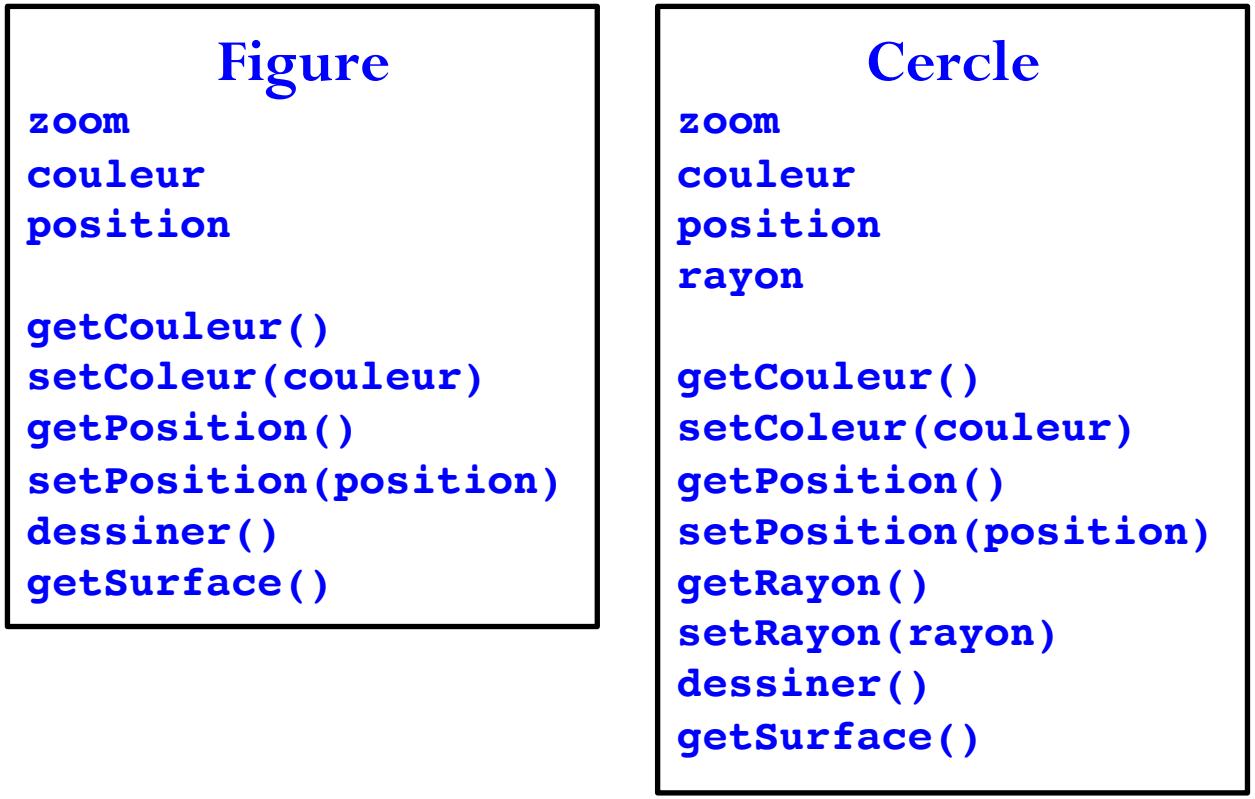



```
public class Cercle extends Figure {
        private int rayon ;
        // Constructeurs
        // Getters/Setters
        @Override
        public void dessiner(Graphics g) {
                g.setColor(couleur);
                g.fillOval(position.getX()-rayon,
                position.getY()-rayon, 2*rayon, 2*rayon);
        }

        @Override
        public double getSurface() {
                return 3.14 * rayon * rayon;
        }
}
```

```
Cercle(rayon)
Cercle(x, y, rayon)
int getRayon()
void setRayon(rayon)
```

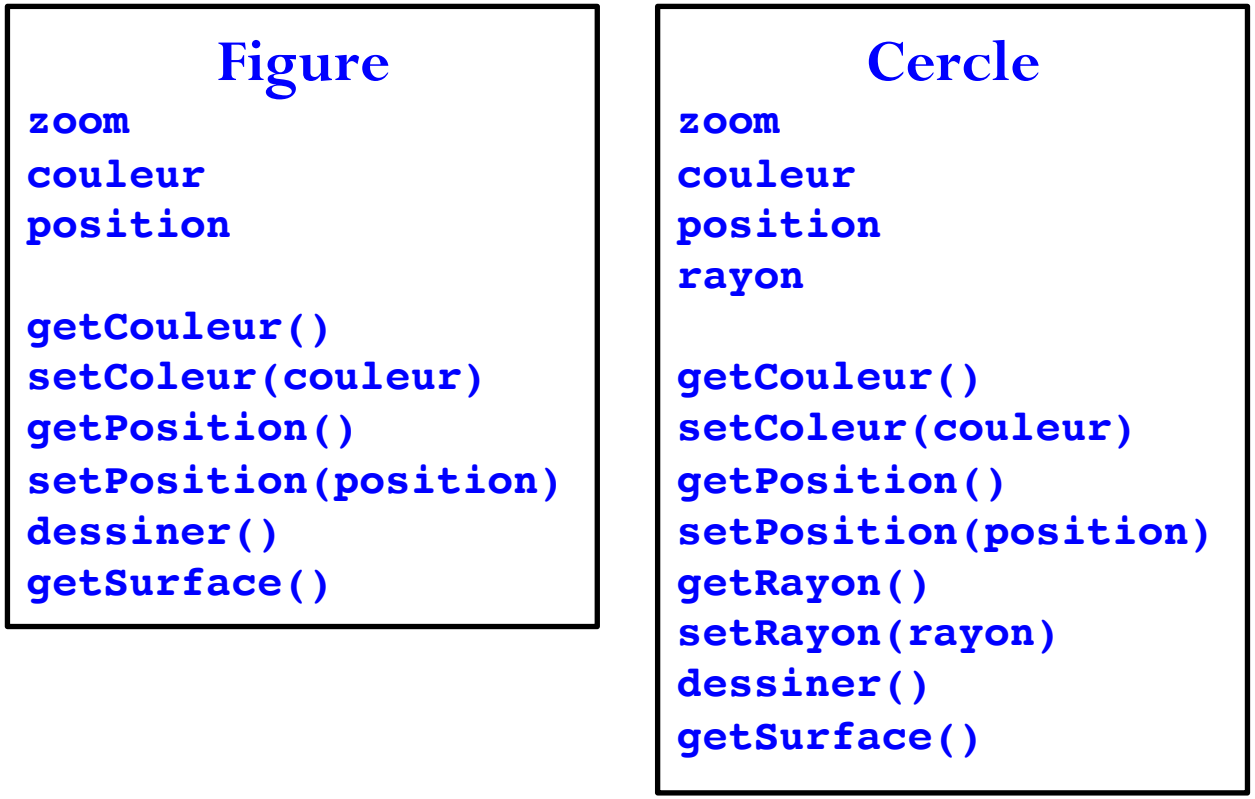
L'HÉRITAGE

- Notion de vue : **`Cercle c1 = new Cercle();`**



L'HÉRITAGE

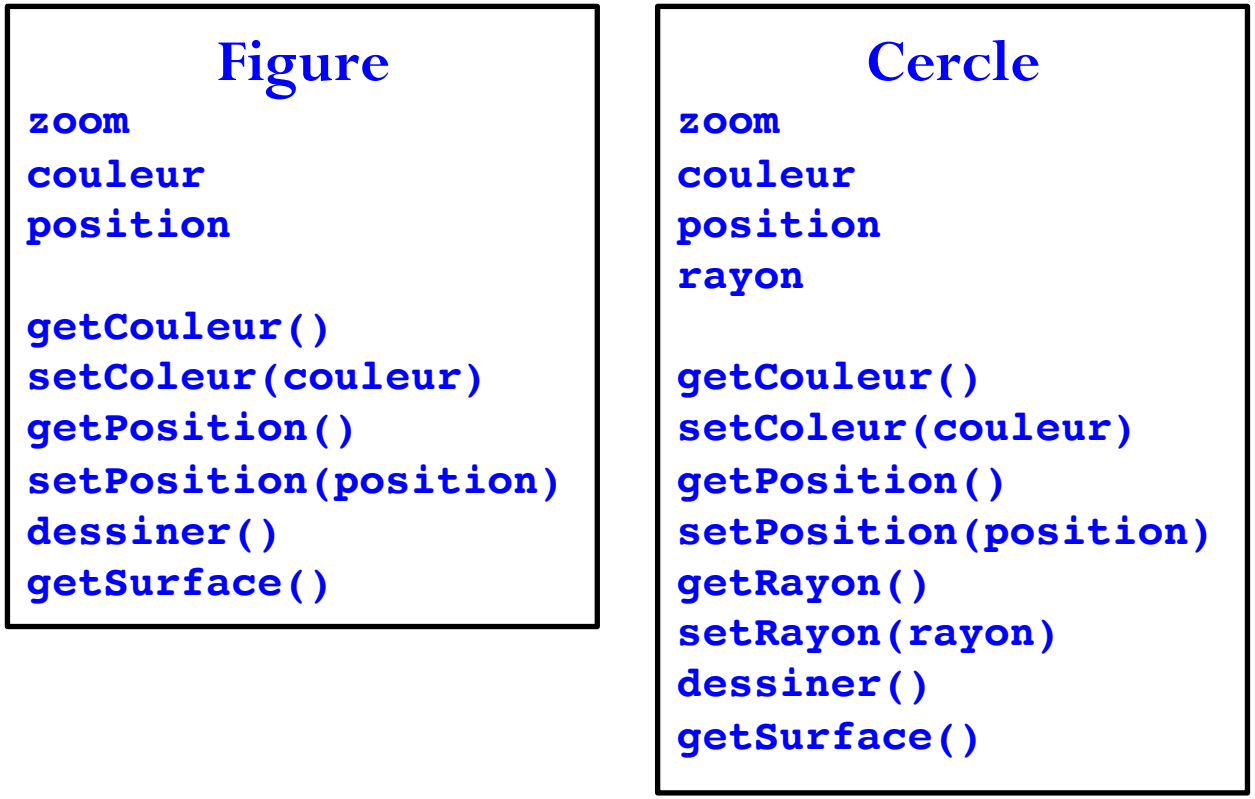
- Notion de vue : **`Figure c1 = new Cercle();`**



On ne peut pas écrire : c1.getRayon()
→ c1 est un cercle qui se voit comme une figure.

L'HÉRITAGE

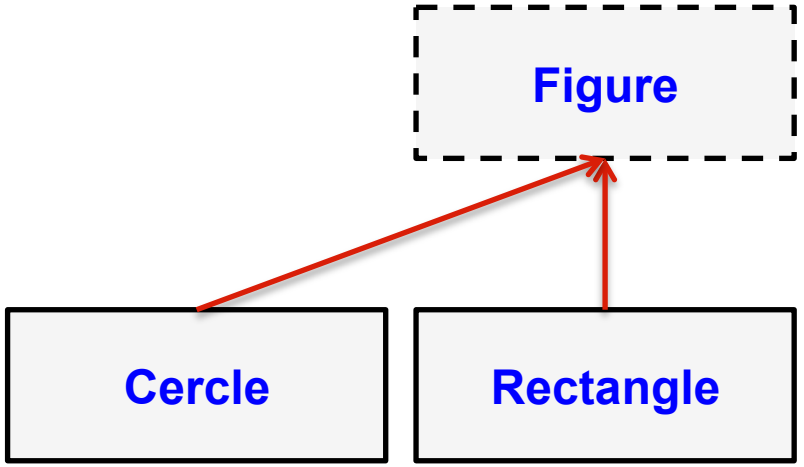
- Notion de Cast : **`Figure c1 = new Cercle();`**



On peut par contre écrire : ((Cercle) c1).getRayon()

POLYMORPHISME

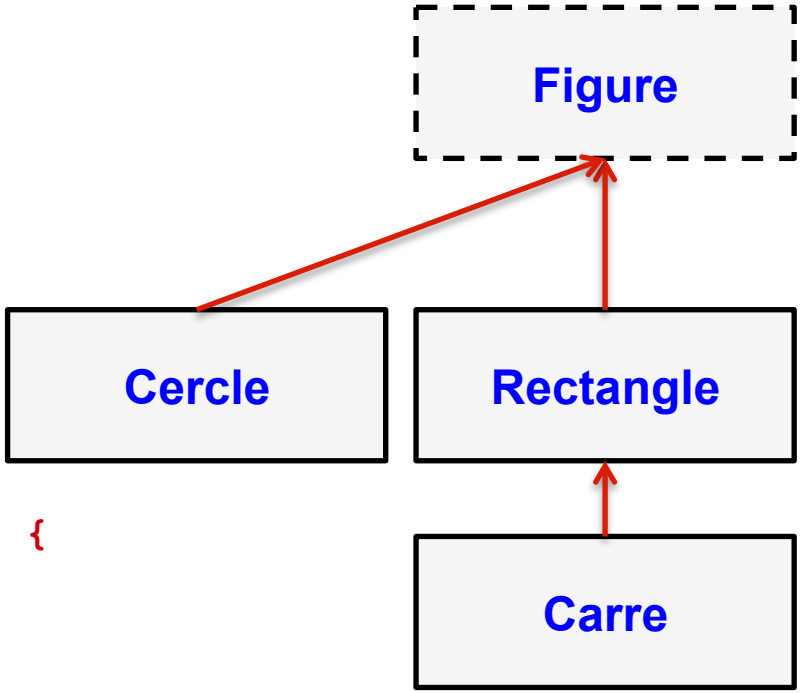
- L'héritage : Classe Rectangle




```
public class Rectangle extends Figure {
        protected int a ;
        protected int b ;
        // Getters/Setters

        @Override
        public void dessiner(Graphics g) {
                g.setColor(couleur);
                g.fillRect(position.getX()-a/2, position.getY()-b/2, a, b);
        }

        @Override
        public double getSurface() {
                return a*b;
        }

}
```

POLYMORPHISME

- L'héritage : Classe Carre




```
public class Carre extends Rectangle {
        protected int v ;
        // Getters/Setters
        public Carre(int x, int y, int v) {
                super(x, y, v, v);
        }
        @Override
        public void setA(int v) {
                a = v;
                b = v;
        }
        @Override
        public void setB(int v) {
                a = v;
                b = v;
        }
}
```

POLYMORPHISME

- Exemple :

```
Console :
50.24
120.0
12.56
49.0
40.0
```

```
public class ClassePrincipale {
        public static void main(String[] args) {
                Figure [] tableau = new Figure[5];
                tableau[0] = new Cercle(10, 10, 4);
                tableau[1] = new Rectangle(50, 50, 15, 8);
                tableau[2] = new Cercle(100, 50, 2);
                tableau[3] = new Carre(100, 100, 7);
                tableau[4] = new Rectangle(50, 200, 20, 2);

                for(int i=0; i<tableau.length; i++) {
                        System.out.println(tableau[i].getSurface());
                }
        }
}
```

POLYMORPHISME

- Exemple :

```
Console :
Cercle(10, 10) [50.24]
Rectangle(50, 50) [120.0]
Cercle(100, 50) [12.56]
Carre(100, 100) [49.0]
Rectangle(50, 200) [40.0]
```

```
public class ClassePrincipale {
        public static void main(String[] args) {
                Figure [] tableau = new Figure[5];
                tableau[0] = new Cercle(10, 10, 4);
                tableau[1] = new Rectangle(50, 50, 15, 8);
                tableau[2] = new Cercle(100, 50, 2);
                tableau[3] = new Carre(100, 100, 7);
                tableau[4] = new Rectangle(50, 200, 20, 2);

                for(int i=0; i<tableau.length; i++) {
                        System.out.println(tableau[i]);
                }
        }
}
```

POLYMORPHISME

- Exemple :

```
public class ClassePrincipale {
        public static void main(String[] args) {
                Figure [] tableau = new Figure[5];
                tableau[0] = new Cercle(10, 10, 4);
                tableau[1] = new Rectangle(50, 50, 15, 8);
                tableau[2] = new Cercle(100, 50, 2);
                tableau[3] = new Carre(100, 100, 7);
                tableau[4] = new Rectangle(50, 200, 20, 2);

                System.out.println(Arrays.toString(tableau));
        }
}
```

```
[Cercle(10, 10) [50.24], Rectangle(50, 50) [120.0], Cercle(100, 50) [12.56], Carre(100, 100) [49.0], Rectangle(50, 200) [40.0]]
```

TYPES ÉNUMÉRÉS

- Enumération :

```
public class Figure {
        public static final int NORMAL = 1;
        public static final int GROS = 2;
        public static final int FIN = 3;
        public static final int TIRET = 4;

        private int type = Figure.NORMAL ;
}
```

POLYMORPHISME

- L'interface Comparable :

```
public class Figure implements Comparable<Figure> {
        ...

        @Override
        public int compareTo(Figure figure) {
                if(getSurface()>figure.getSurface())
                        return 1;
                else
                        if(getSurface()<figure.getSurface())
                                return -1;
                        else
                                return 0;
        }

}
```

GÉNÉRICITÉ

```
public class ClasseGenerique<T> {

        private T v ;

        public void affecter(T v) {
                this.v = v ;
        }

        public void afficer() {
                System.out.println(v);
        }

}
```

Lorsque l'on utiliser un type qui devrait être définit par l'utilisateur
Exemple : Tableau (si on développer un objet Tableau, le contenu du tableau doit être de type générique.

GÉNÉRICITÉ

```
public class Main {

        public static void main(String[] args) {
                ClasseGenerique<String> cg = new ClasseGenerique<String>();
                cg.affecter("Bonjour");
                cg.afficer();
        }

}
```

GÉNÉRICITÉ

```
import java.util.LinkedList;
public class ClasseGenerique<T> {

        private LinkedList<T> liste ;

        public ClasseGenerique() {
                liste = new LinkedList<T>();
        }

        public void ajouter(T v) {
                liste.add(v);
        }

        public void afficer() {
                for(T v : liste) {
                        System.out.println(v);
                }
        }
}
```

GÉNÉRICITÉ

```
public class Main {

        public static void main(String[] args) {
                ClasseGenerique<String> cg = new ClasseGenerique<String>();
                cg.ajouter("Bonjour");
                cg.ajouter("Salut");
                cg.ajouter("Au revoir");
                cg.afficer();
        }

}
```

TABLEAUX DYNAMIQUES (COLLECTIONS)

- Tableaux dynamiques
  - Vector (add, add(i), remove(objet), remove(i), contains, …)
  - ArrayList (add, add(i), remove(objet), remove(i), contains, …)
- Listes chainées
  - LinkedList (add, add(i), addFirst, addLast, removeFirst, remove(i), removeLast, …)
- Piles
  - Stack (push, pop, pull, …)
- Ensembles (pas de doublons)
  - HashSet
- Collection (Clé, Valeur)
  - HashMap (add(clé, valeur), remove(clé), …)
- … (TreeSet, ...)

→ foreach / iterator / thread safe

TABLEAUX DYNAMIQUES

- HashMap

```
HashMap<String, String> hm = new HashMap<String, String>();

hm.put("a", "Université");
hm.put("b", "Faculté");
hm.put("c", "Département");
hm.put("d", "Master");

hm.remove("d");

System.out.println(hm.get("b"));
```

TABLEAUX DYNAMIQUES

- Tri : plusieurs techniques, dont :

```
Collections.sort(nom_collection);
```

TABLEAUX DYNAMIQUES

- Tri des figures : (dans le main)

```
Vector<Figure> tableau = new Vector<Figure>();
tableau.add(new Cercle(10, 10, 4));
tableau.add(new Rectangle(50, 50, 15, 8));
tableau.add(new Cercle(100, 50, 2));
tableau.add(new Carre(100, 100, 7));
tableau.add(new Rectangle(50, 200, 20, 2));

for(int i=0; i<tableau.size(); i++) {
        System.out.println(tableau.get(i));
}

Collections.sort(tableau);

System.out.println();
for(int i=0; i<tableau.size(); i++) {
        System.out.println(tableau.get(i));
}
```

Console

```
Cercle(10, 10) [50.24]
Rectangle(50, 50) [120.0]
Cercle(100, 50) [12.56]
Carre(100, 100) [49.0]
Rectangle(50, 200) [40.0]

Cercle(100, 50) [12.56]
Rectangle(50, 200) [40.0]
Carre(100, 100) [49.0]
Cercle(10, 10) [50.24]
Rectangle(50, 50) [120.0]
```

Comment fait-il que les figures sont triées à base de leurs surfaces ?

THREADS

- Par héritage (Compteur)

```
public class Compteur extends Thread {
        @Override
        public void run() {
                for(int i=0; i<10; i++) {
                        System.out.println(i);
                        try {
                                sleep(1000);
                        } catch (InterruptedException e) {}
                }
        }
}
```

THREADS

- Par héritage (Main)

```
public class ClassePrincipale {
        public static void main(String[] args) {
                Compteur c1 = new Compteur();
                Compteur c2 = new Compteur();
                c1.start();
                c2.start();
        }
}
```

Console

```
0
0
1
1
2
2
3
3
4
4
5
5
6
6
7
7
8
8
9
9
```

THREADS

- Par implémentation (Compteur)

```
public class Compteur implements Runnable {
        @Override
        public void run() {
                for(int i=0; i<10; i++) {
                        System.out.println(i);
                        try {
                                Thread.sleep(1000);
                        } catch (InterruptedException e) {}
                }
        }
}
```

THREADS

- Par implémentation (Main)

```
public class ClassePrincipale {
        public static void main(String[] args) {
                Thread c1 = new Thread(new Compteur());
                Thread c2 = new Thread(new Compteur());
                c1.start();
                c2.start();
        }
}
```

Console

```
0
0
1
1
2
2
3
3
4
4
5
5
6
6
7
7
8
8
9
9
```

THREADS

- Lancer une action dans un Thread :

```
public class ClassePrincipale {
        public static void main(String[] args) {
                new Thread(new Runnable() {
                        @Override
                        public void run() {
                                for(int i=0; i<10; i++) {
                                        try {
                                                Thread.sleep(1000);
                                        } catch (InterruptedException e) {}
                                        System.out.println(i);
                                }
                        }
                }).start();
                System.out.println("Salut");
        }
}
```

THREADS : SYNCHRONISATION

- Regardez bien cet exemple :
  - Un boulanger ayant 100 baguettes et qui donne le nombre de baguettes lui restantes chaque seconde.

```
public class Boulanger extends Thread {
        private int n = 100;
        public void run() {
                while(true) {
                        try {
                                System.out.println(n);
                                Thread.sleep(1000);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }

        public void prendre() {
                n--;
        }
}
```

THREADS : SYNCHRONISATION

- Regardez bien cet exemple :
  - Un client qui va acheter chaque 5 secondes une baguette.

```
public class Client extends Thread {

        private Boulanger boulanger ;

        public Client(Boulanger boulanger) {
                this.boulanger = boulanger;
        }

        @Override
        public void run() {
                while(true) {
                        try {
                                boulanger.prendre();
                                Thread.sleep(5000);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }

}
```

THREADS : SYNCHRONISATION

- Regardez bien cet exemple :
  - Soit 10 clients qui achètent 1 baguette chacun au même temps chaque 5 secondes.

```
public class ClassePrincipale {

        public static void main(String[] args) {
                new Thread(new Runnable() {
                        @Override
                        public void run() {
                                Boulanger boulanger = new Boulanger();
                                boulanger.start();
                                for(int i=0; i<10; i++) {
                                        new Client(boulanger).start();
                                }
                        }
                }).start();
        }

}
```

THREADS : SYNCHRONISATION

- Regardez bien cet exemple :
  - Exécution ...

	100		57		28
	90		57		28
10	90	*8*	57	*9*	28
	90		57		28
	90		57		28
	80		50		19
10	80	*7*	50	*9*	19
	80		50		19
	80		50		19
	80		50		19
	71		43		11
9	71	*7*	43	*8*	11
	71		43		11
	71		43		11
	71		43		11
	65		37		3
7	65	*6*	37	*8*	3
	65		37		3
	65		37		3
	65		37		3

THREADS : SYNCHRONISATION

- Regardez bien cet exemple :
  - Un boulanger ayant 100 baguettes et qui donne le nombre de baguettes lui restantes chaque seconde.

```
public class Boulanger extends Thread {
        private int n = 100;
        public void run() {
                while(true) {
                        try {
                                System.out.println(n);
                                Thread.sleep(1000);
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }

        public synchronized void prendre() {
                n--;
        }
}
```

THREADS : SYNCHRONISATION

- Regardez bien cet exemple :
  - Exécution ...

```
100     50      10
90      50      10
90      50      10
90      50      10
90      50      10
80      40      0
80      40      0
80      40      0
80      40      0
80      40      0
70      30
70      30
70      30
70      30
70      30
60      20
60      20
60      20
60      20
60      20
```

LES FLUX

- Octets :
  - InputStream in; (read, read(b), read(b, o, l))
  - OutputStream out; (write, write(b), write(b, o, l))
- Caractères :
  - Reader r;
  - Writer w; (write(String))
- Passer d'octers à caractères :
  - InputStreamReader(flux_en_octets)
  - OutputStreamWriter(flux_en_octets)
- flush() et close()
- Exemples :
  - FileInputStream f ou FileOutputStream()
  - InputStream in = System.in
  - OutputStream out = System.out

Réseau et Sockets

- HTTP (lecture)

```
URL url = new URL("http://...");

InputStream is = url.openStream();

... is.read()
```

Réseau et Sockets

- HTTP (lecture)

```
URL url = new URL("http://...");

URLConnection connection = url.openConnection();

InputStream is = connection.getInputStream();

... is.read()
```

Réseau et Sockets

- HTTP (ecriture)

```
URL url = new URL("http://...");
URLConnection connection = url.openConnection();
connection.setDoOutput(true);

OutputStream out = connection.getOutputStream();
out.write('x');
out.write('=');
out.write('3');

out.flush();
out.close();
```

Réseau et Sockets

- HTTP (ecriture)

```
URL url = new URL("http://...");
URLConnection connection = url.openConnection();
connection.setDoOutput(true);

OutputStream out = connection.getOutputStream();
out.write('x');
out.write('=');
out.write('3');

InputStream is = connection.getInputStream();
System.out.println((char) is.read());

out.flush();
out.close();
```

Réseau et Sockets

- HTTP (headers) :

```
URL url = new URL("http://...");

URLConnection connection = url.openConnection();

connection.getHeaderFields();

ou

System.out.println(connection.getHeaderField("Content-Length"));
System.out.println(connection.getHeaderField(2));
```

Réseau et Sockets

- Serveur :

int port = 3000;

```
ServerSocket ss = new ServerSocket(port)
Socket s = ss.accept();
InputStream in = s.getInputStream();
OutputStream in = s.getOutputStream();
```

Réseau et Sockets

- Client :

String host = "localhost";

int port = 3000;

```
Socket s = new Socket(host, port);
InputStream in = s.getInputStream();
OutputStream in = s.getOutputStream();
```

COMPOSANTS GRAPHIQUES ET IHM

- Fenêtre :

```
public class ClassePrincipale {
          public static void main(String[] args) {
                    JFrame fenetre = new JFrame("Mon Application");
                    fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                    fenetre.setBounds(100, 100, 800, 600);
                    fenetre.setVisible(true);
          }
}
```

COMPOSANTS GRAPHIQUES ET IHM

- Panneau :

```
public class ClassePrincipale {
          public static void main(String[] args) {
                    JFrame fenetre = new JFrame("Mon Application");


                    JPanel panneau = new JPanel();
                    fenetre.add(panneau);
                    fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                    fenetre.setBounds(100, 100, 800, 600);
                    fenetre.setVisible(true);
          }
}
```

COMPOSANTS GRAPHIQUES ET IHM

- Texte :

```
public class ClassePrincipale {
          public static void main(String[] args) {
                    JFrame fenetre = new JFrame("Mon Application");
                    JLabel texte = new JLabel("Bonjour à tous");
                    panneau.add(texte);
                    JPanel panneau = new JPanel();
                    fenetre.add(panneau);
                    fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                    fenetre.setBounds(100, 100, 800, 600);
                    fenetre.setVisible(true);
          }
}
```

COMPOSANTS GRAPHIQUES ET IHM

- Bouton :

```
public class ClassePrincipale {
        public static void main(String[] args) {
                JFrame fenetre = new JFrame("Mon Application");
                JLabel texte = new JLabel("Bonjour à tous");
                panneau.add(texte);
                JPanel panneau = new JPanel();
                fenetre.add(panneau);
                JButton bouton = new JButton("OK");
                bouton.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                                texte.setText("Au-revoir tout le monde");
                        }
                });
                panneau.add(bouton);
                fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                fenetre.setBounds(100, 100, 800, 600);
                fenetre.setVisible(true);
        }
}
```

COMPOSANTS GRAPHIQUES ET IHM

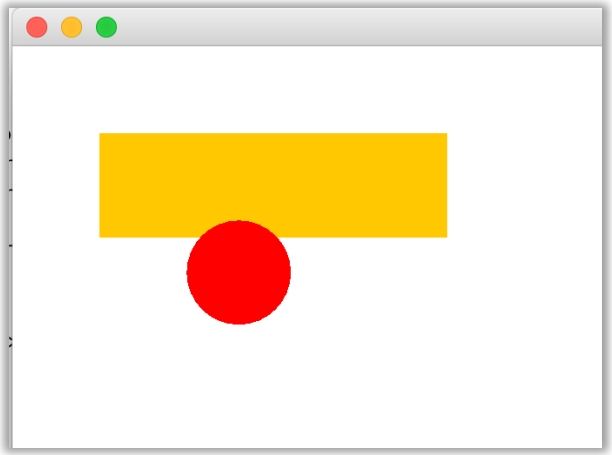
- Ajouter son propre dessin :

```
public class MaFeuille extends JComponent {

        @Override
        public void paintComponent(Graphics g) {
                super.paintComponent(g);
                g.setColor(Color.ORANGE);
                g.fillRect(50, 50, 200, 60);
                g.setColor(Color.RED);
                g.fillOval(100, 100, 60, 60);
        }

}
```

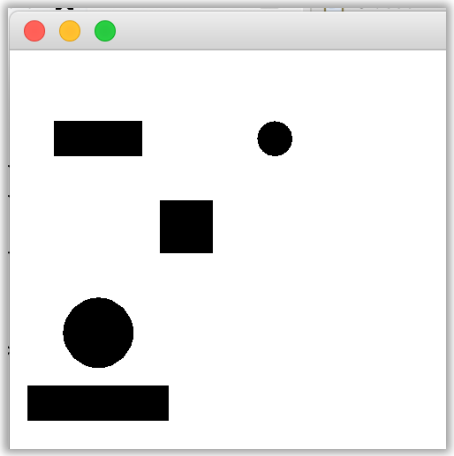
COMPOSANTS GRAPHIQUES ET IHM

- Dessiner sur un panneau :

```
public class ClassePrincipale {
        public static void main(String[] args) {
                JFrame fenetre = new JFrame("Mon Application");
                fenetre.getContentPane().setBackground(Color.WHITE);
                MaFeuille feuille= new MaFeuille();
                fenetre.add(feuille);
                fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                fenetre.setBounds(100, 100, 800, 600);
                fenetre.setVisible(true);
        }
}
```



COMPOSANTS GRAPHIQUES ET IHM



- Dessinons nos figures dans MaFeuille :

```
public class MaFeuille extends JComponent {
        private Vector<Figure> tableau ;
        public MaFeuille() {
                tableau = new Vector<Figure>();
                tableau.add(new Cercle(50, 160, 40));
                tableau.add(new Rectangle(50, 50, 50, 20));
                tableau.add(new Cercle(150, 50, 20));
                tableau.add(new Carre(100, 100, 30));
                tableau.add(new Rectangle(50, 200, 80, 20));
        }
        @Override
        public void paintComponent(Graphics g) {
                super.paintComponent(g);
                for(int i=0; i<tableau.size(); i++) {
                        tableau.get(i).dessiner(g);
                }
        }
}
```

COMPOSANTS GRAPHIQUES ET IHM

- Evènements sur un composant (souris ou clavier) :

```
public class MaFeuille extends JComponent implements MouseListener {

        public MaFeuille() {
                addMouseListener(this);
        }

        @Override
        public void mouseClicked(MouseEvent e) {
                System.out.println(e.getX()+" "+e.getY());
        }

        ...

}
```

COMPOSANTS GRAPHIQUES ET IHM

- Ajoutons un cercle dans l'endroit où l'on a cliqué

```
public class MaFeuille extends JComponent implements MouseListener {
        private Vector<Figure> tableau ;
        public MaFeuille() {
                addMouseListener(this);
                tableau = new Vector<Figure>();
                tableau.add(new Cercle(50, 160, 40));
                ...
        }

        @Override
        public void paintComponent(Graphics g) {...}

        @Override
        public void mouseClicked(MouseEvent e) {
                tableau.add(new Cercle(e.getX(), e.getY(), 20));
                repaint();
        }
}
```

Petit Exemple Sympa (IHM)

UNE FENÊTRE

1. Je crée une **Fenêtre** simple (héritant de JFrame) :

```
public class Fenetre extends JFrame {

        public Fenetre() {
                super();
                setTitle("Mon Application");
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setBounds(100, 100, 800, 600);
                setVisible(true);
        }

}
```

UN PLATEAU

2. Je crée un objet **Plateau** (héritant de JPanel) :

```
public class Plateau extends JPanel {

        public Plateau() {
                super();
                setBackground(Color.WHITE);
        }

}
```

UN SOMMET CARRÉ

3. Je crée un objet **Sommet** carré de couleur rouge (héritant de JPanel) :

```
public class SommetR extends JPanel {

        public SommetR() {
                super();
                setBackground(Color.RED);
        }

}
```

AJOUTER LE PLATEAU À LA FENÊTRE

4. J'ajoute le plateau à la fenêtre :

```
public class Fenetre extends JFrame {

        public Fenetre() {
                Plateau plateau = new Plateau();
                add(plateau);
                setTitle("Mon Application");
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setBounds(100, 100, 800, 600);
                setVisible(true);
        }

}
```

AJOUTER UN ÉCOUTEUR DE CLIC AU PLATEAU

5. J'ajoute un écouteur de clic au plateau :

```
public class Plateau extends JPanel implements MouseListener {

        public Plateau() {
                super();
                setBackground(Color.WHITE);
                addMouseListener(this);
        }

}
```

AJOUTER UN SOMMET À L'ENDROIT CLIQUÉ

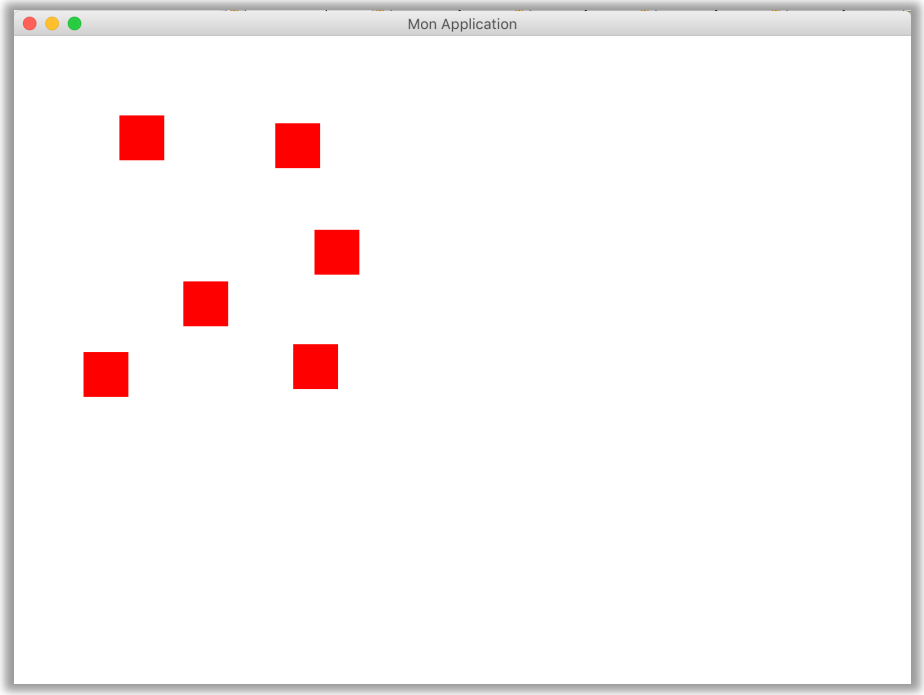
6. A chaque clic sur le plateau on ajoute un Sommet à l'endroit cliqué :

```
public class Plateau extends JPanel implements MouseListener {

        public Plateau() {
                setBackground(Color.WHITE);
                addMouseListener(this);
                setLayout(null);
        }

        @Override
        public void mouseClicked(MouseEvent e) {
                Sommet sommet = new Sommet();
                sommet.setBounds(e.getX()-20, e.getY()-20, 40, 40);
                add(sommet);
                repaint();
        }
}
```

TEST

7. Testez :

```
public class ClassePrincipale {
        public static void main(String[] args) {
                Fenetre fenetre = new Fenetre();
        }
}
```



AJOUTER UN SOMMET SOUS FORME DE DISQUE

8. Transformer la forme du sommet en un disque :

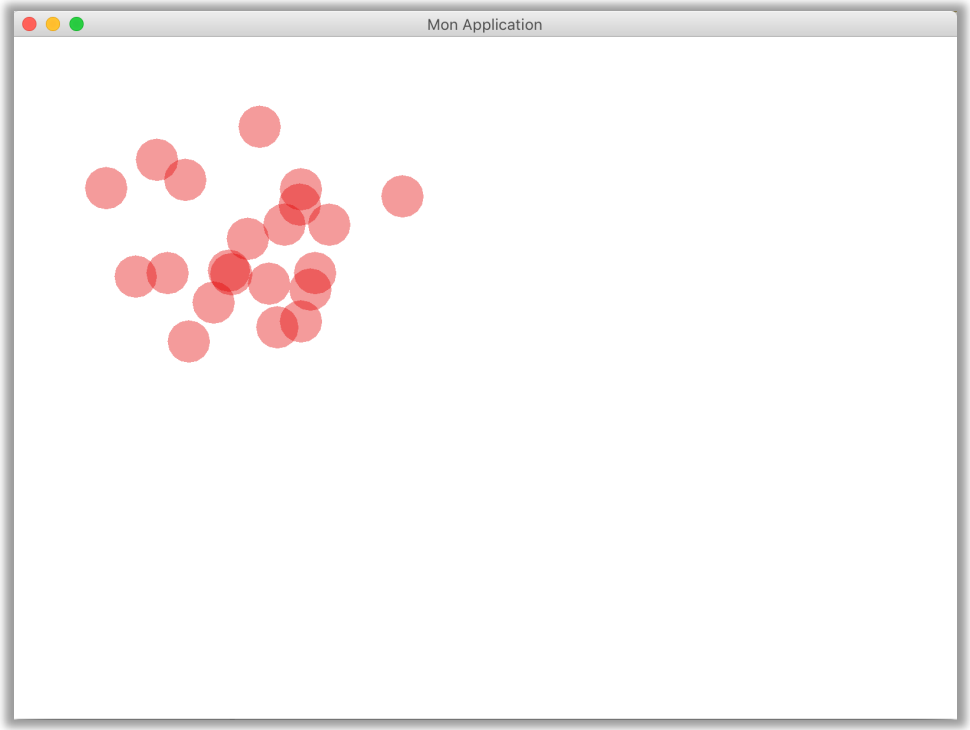
```
public class Sommet extends JComponent {

        public Sommet() {
                super();
        }

        @Override
        public void paintComponent(Graphics g) {
                super.paintComponent(g);
                g.setColor(new Color(230, 0, 0, 100));
                g.fillOval(2, 2, getWidth()-4, getHeight()-4);
        }
}
```

Dessine un disque rouge transparent

UNE FENÊTRE

9. Testez :

```
public class ClassePrincipale {
        public static void main(String[] args) {
                Fenetre fenetre = new Fenetre();
        }
}
```



UNE FENÊTRE

10. Déplacez les sommets :

```
public class Sommet extends JComponent implements MouseMotionListener {

        public Sommet() {
                super();
                addMouseMotionListener(this);
        }

        @Override
        public void paintComponent(Graphics g) { ... }

        @Override
        public void mouseDragged(MouseEvent e) {
                setBounds(e.getX()+getX()-20, e.getY()+getY()-20,
                getWidth(), getHeight());
        }
}
```

UNE FENÊTRE

11. Testez :

```
public class ClassePrincipale {
        public static void main(String[] args) {
                Fenetre fenetre = new Fenetre();
        }
}
```

UNE FENÊTRE

12. Plus pro !

```
public class Sommet extends JComponent implements MouseMotionListener {
        private int dx, dy;
        …
        @Override
        public void mousePressed(MouseEvent e) {
                dx = getWidth()/2-e.getX();
                dy = getHeight()/2-e.getY();
        }

        @Override
        public void mouseDragged(MouseEvent e) {
                setBounds(
                e.getX()+getX()-getWidth() /2 + dx,
                e.getY()+getY()-getHeight()/2 + dy,
                getWidth(), getHeight());
        }
}
```

Merci de Votre Attention