

Les concepts de l'orienté-objet

.. un survol par l'exemple

Orienté-objet

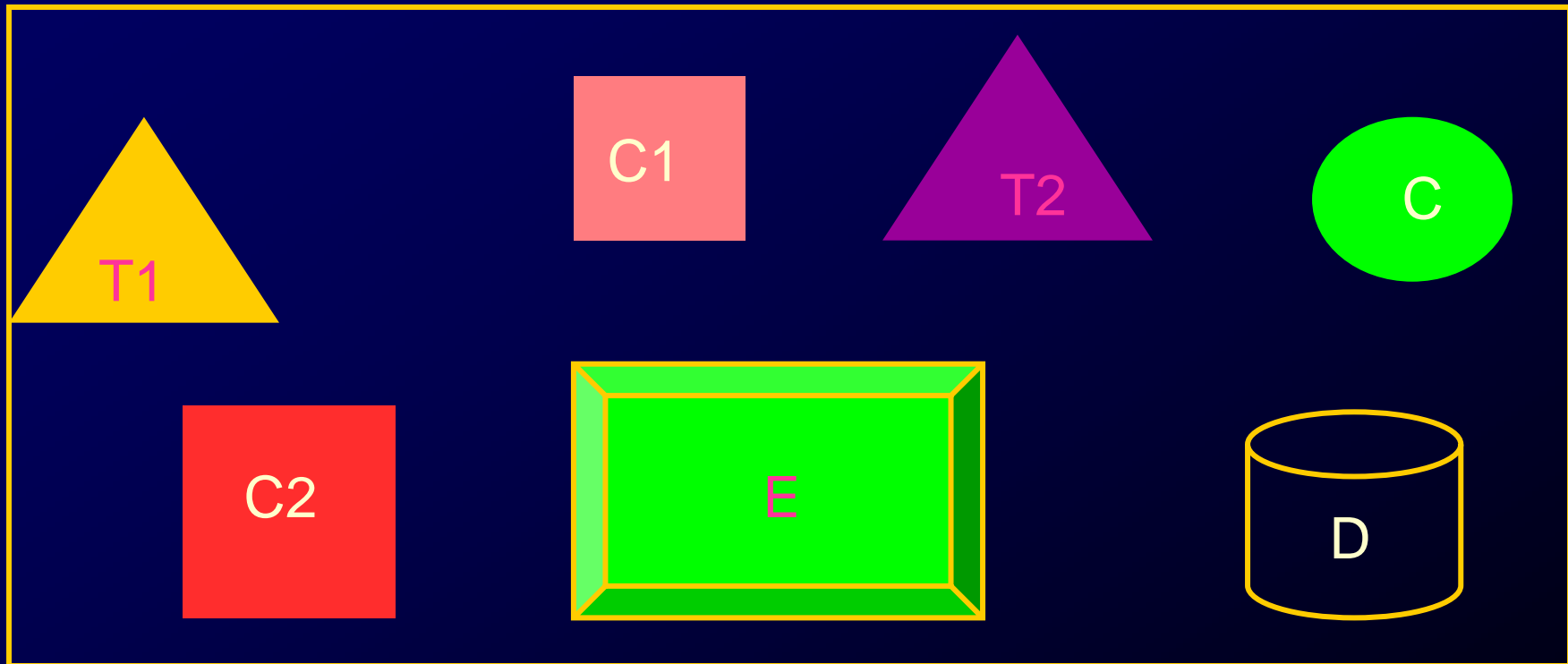
- ◆ Méthode de conception qui permet de programmer en terme d'*objets*
 - La notion d'objet permet une représentation **abstraite** du système proche du monde réel
 - Modèle
 - ensemble d'objets représentants des entités du monde réel
 - Objet = données + opérations
 - informations associées à l'objet
 - ensemble d'opérations (comportement)

Orienté-objet

- Programme
 - Ensemble d'objets qui communiquent à l'aide de messages
- Penser *d'abord* aux types de données et *ensuite* aux opérations qu'ils peuvent fournir
- Java, C++, Eiffel, SmallTalk

Exemple

- ◆ Application graphique qui gère les formes



Objets

- ◆ Chaque objet possède des **informations** qui lui sont associées
- ◆ Chaque objet possède une **responsabilité**
 - Il fournit un **service** qui est utile à d'autres membres de la communauté
 - Application graphique
 - *L'objet E est responsable de l'affichage des données sur un écran du système*
 - *L'objet C est responsable de fournir les informations pour le cercle qu'il représente (p.ex: la taille, les coordonnées)*

Attributs

- ◆ Objet: **Données** + Opérations
 - Attribut: déclaration d'une *variable interne* à l'objet
 - L'ensemble des attributs forme l'état de l'objet
 - Les valeurs des attributs peuvent changer pendant l'exécution
 - Un attribut représente un autre objet

Attributs

◆ Application graphique

- *L 'objet C a un attribut **rayon** qui détermine la taille du cercle et un attribut **couleur** qui détermine sa couleur.*
- *Le rayon est fixe pendant toute la durée de vie de l'objet, mais sa couleur peut changer à l'affichage*

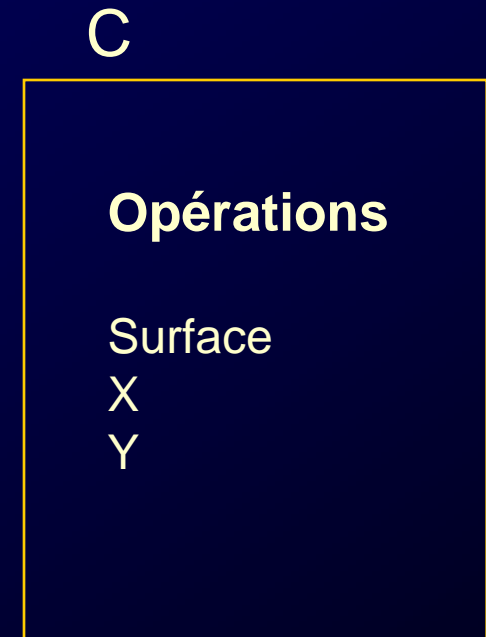
C

Attributs

rayon
couleur
x
y

Méthodes

- ◆ **Objet: Données + Opérations**
- ◆ **Méthode:**
 - opération qui permet d'accéder ou de modifier les données de l'objet
 - opération qui réalise un certain comportement
 - Application graphique :
 - *L'objet E possède une méthode pour afficher une forme sur l'écran du système*
 - *L'objet C possède une méthode qui calcule sa propre surface*



Encapsulation

- ◆ *Partie visible* d'un objet
 - Opérations
- ◆ *Partie invisible* d'un objet
 - données
 - description du comportement des opérations

C

Opérations

Surface
X
Y

Description

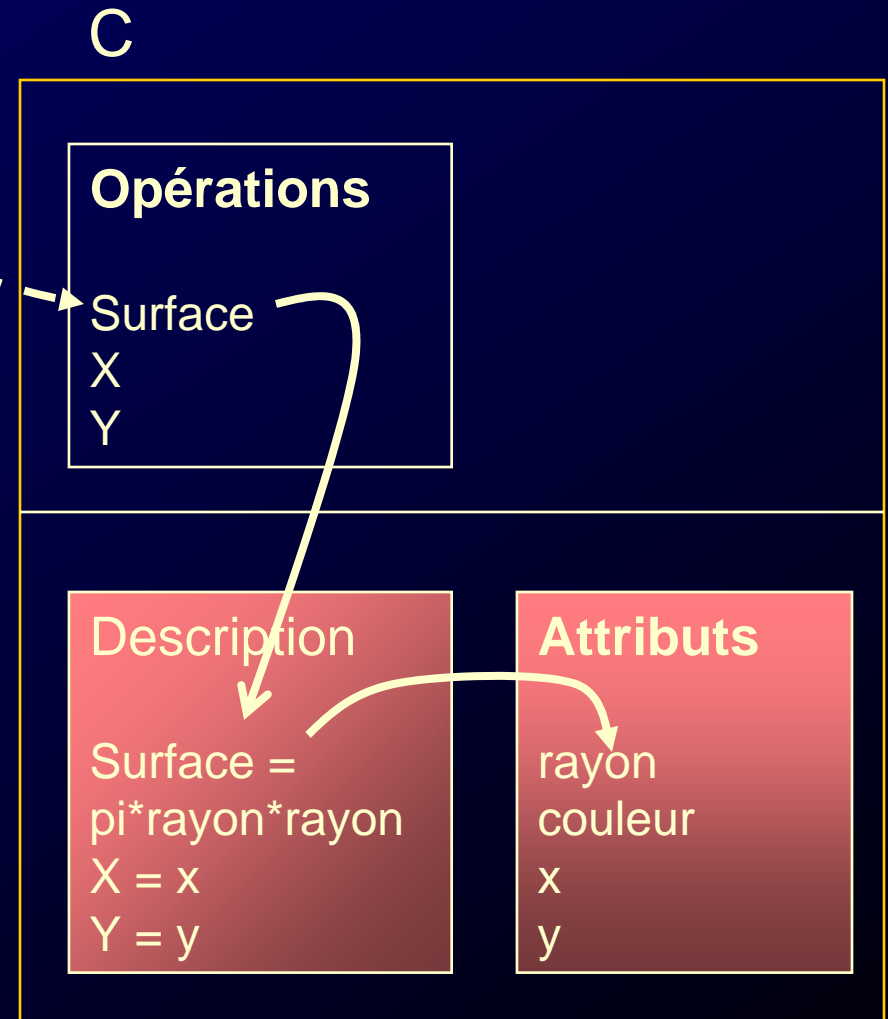
Surface =
 $\pi \times \text{rayon} \times \text{rayon}$
 $X = x$
 $Y = y$

Attributs

rayon
couleur
x
y

Encapsulation

- Manipulation de l'information cachée n'a lieu que lorsqu'un autre objet **ordonne** à l'objet d'exécuter une opération
- Pourquoi l'encapsulation?
 - Lorsqu'on utilise un objet on a besoin de savoir:
 - ce qu'il sait faire mais ...
 - pas comment il sait le faire



Type de données abstrait

- Un type de données abstrait est un modèle équipé d'un certain nombre d'opérations qui affectent le modèle
 - définit ce que l'objet peut faire, mais pas comment il peut le faire
 - La partie visible de l'objet définit le type abstrait
- Application graphique :
 - *Le type Cercle est défini par trois opérations: Surface, qui calcule la taille; X, et Y, qui retournent les coordonnées du centre*

Type Cercle

Opérations

Surface

X

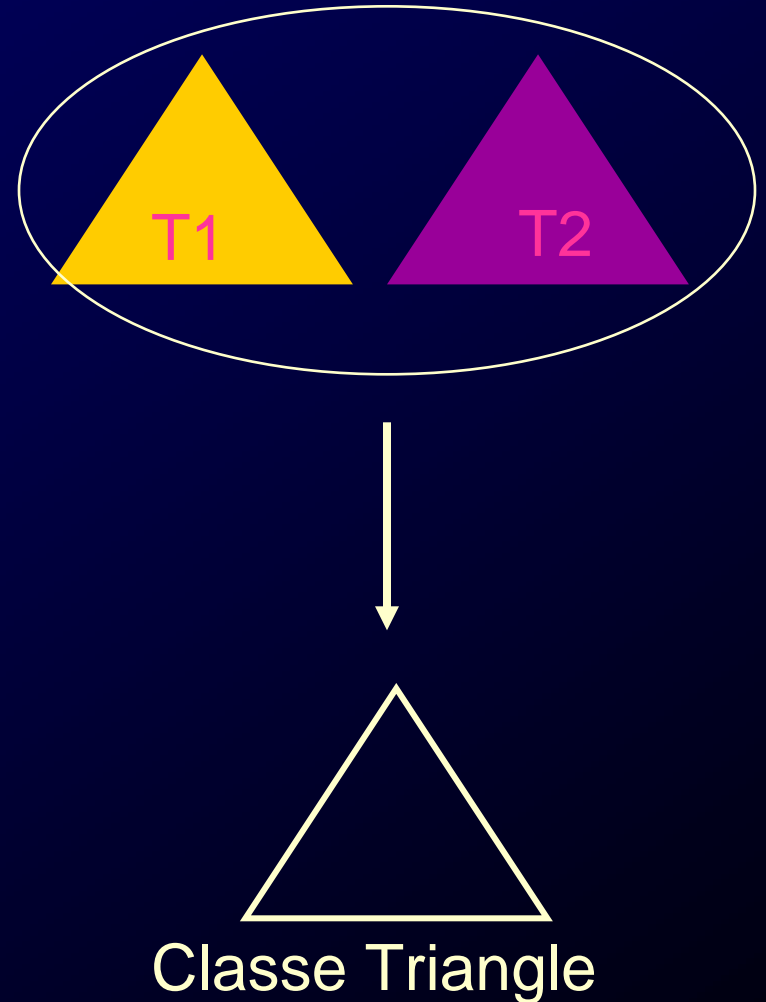
Y

Classes

- ◆ Les objets avec des caractéristiques communes sont groupés en classes
- ◆ Une *classe* définit un moule d'objets
 - Elle décrit les *données* participant à l'état de l'objet
 - Elle précise les *méthodes* de l'objet (« quoi »)
 - Elle décrit le *comportement* des méthodes (« comment »)
 - Mêmes attributs, mêmes méthodes, même comportement
- ◆ Une classe définit *plus* qu'un type

Classes

- Application graphique :
 - *Un programmeur doit fournir des classes pour décrire les triangles, cercles, carrés, ainsi que des classes pour l'écran et le disque.*
 - *La classe Cercle définit le rayon, les coordonnées, la couleur, et la méthode qui calcule et retourne la surface du cercle*

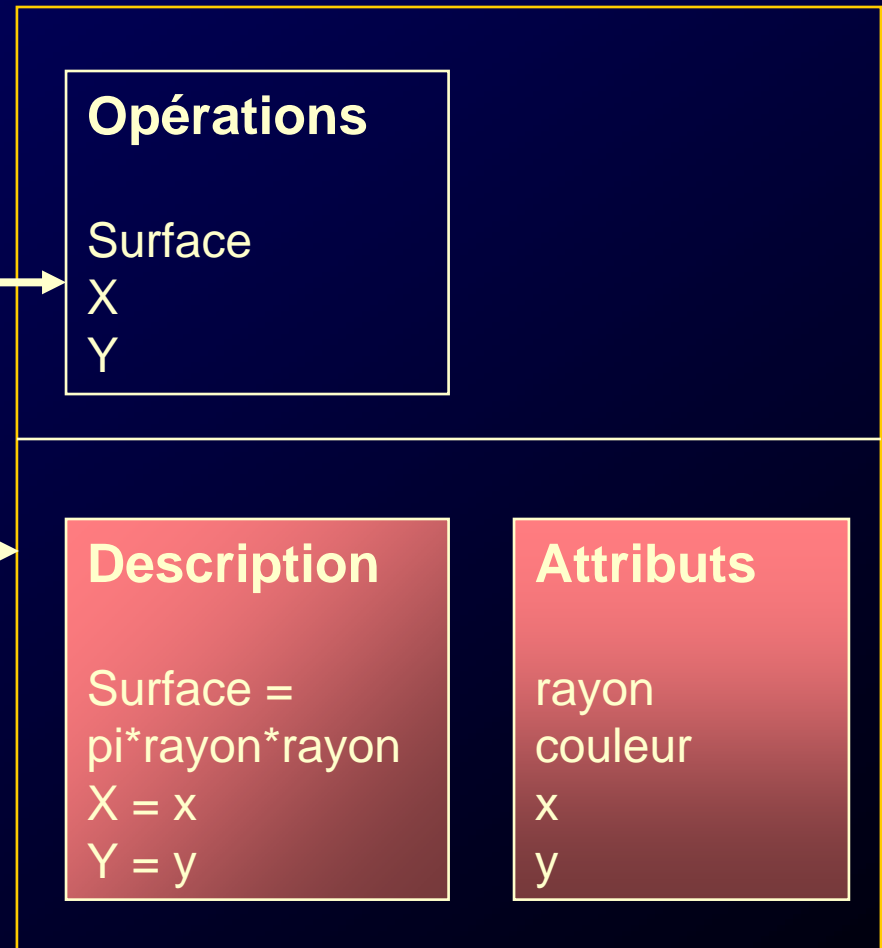


Classes

◆ La classe Cercle:

- Définit la notion du cercle
- Un ensemble visible de méthodes (*l'interface*)
- Dans la partie cachée (*corps*)
 - Décrit le comportement des méthodes
 - Un ensemble d'attributs
 - L'état caché de l'objet qui persiste et qui est utilisé par les méthodes

Classe Cercle



Une classe en Java

```
class Cercle {  
    int x, y; float rayon; /* attributs/état caché des autres classes */  
    public Cercle(int a, int b, float r){ /* Exécuté lorsque l'objet est créé */  
        x = a; y = b; rayon = r ;  
    }  
    public int X() { /* méthode pour rendre la coordonnée x */  
        return x; }  
  
    public int Y() { /* méthode pour rendre la coordonnée y */  
        return y; }  
  
    public float Surface() { /* méthode pour calculer la surface du cercle */  
        return 3.14*rayon*rayon; }  
}
```

Une classe en Java

```
class Cercle {  
    int x, y; float rayon; /* attributs/état caché des autres classes */  
    public Cercle(int a, int b, float r){ /* Exécuté lorsque l'objet est créé */  
        x = a;  b = y;  rayon = r;  
    }  
    public int X() { /* méthode pour rendre la coordonnée x */  
        return x; }  
    public int Y() { /* méthode pour rendre la coordonnée y */  
        return y; }  
    public float Surface() { /* méthode pour calculer la surface du cercle */  
        return 3.14*rayon*rayon; }  
}
```

Interface

Une classe en Java

```
class Cercle {  
    int x, y; float rayon; /* état est caché des autres classes */  
    public Cercle(int a, int b, float r){ /* Exécuté lorsque l'objet est créé */  
        x = a;  b = y;  rayon = r ;  
    }  
  
    public int X() { /* méthode pour rendre la coordonnée x */  
        return x; }  
  
    public int Y() { /* méthode pour rendre la coordonnée y */  
        return y; }  
  
    public float Surface() { /* méthode pour calculer la surface du cercle */  
        return 3.14*rayon*rayon;  
    }  
}
```

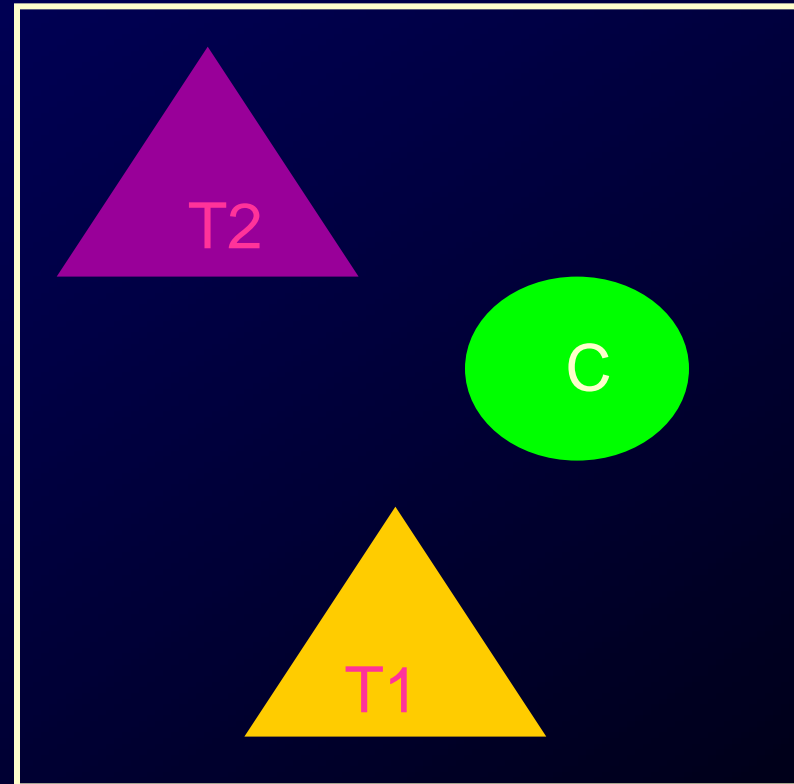
Corps

Programmation vs Exécution

Programmation

```
class Cercle { ... }  
class Triangle {...}  
main(){  
  C = new Cercle(3,4,5.0);  
  T1 = new Triangle(5,3,2.0);  
  T2 = new Triangle(1,2,2.0);  
  ... }
```

Exécution



Un objet n'existe qu'à l'exécution de l'application

Instances

- ◆ Un **objet** est un exemplaire ou une instance d'une classe
 - Tout objet appartient à une classe
 - Chaque objet possède un nom unique
 - Un objet vit indépendamment des autres objets
 - Etat (individuel) qui tient compte de l'effet des opérations
 - Un objet peut changer son état propre
 - Différentes séquences d'appels d'opérations produisent différents états internes

Instances

- ◆ Une application peut avoir plusieurs objets de la même classe
 - Application graphique : les 2 objets de la classe carré sont deux instances de la *même* classe et possèdent donc les mêmes méthodes et types d'attributs, mais ...
 - ils peuvent varier en taille, couleur et positionnement sur l'écran
 - ils peuvent répondre différemment aux messages qu'on leur envoie



Messages - Stimulus

- ◆ Chaque objet comprend un certain nombre de **messages** ou **stimulus** envoyés par d'autres objets
 - pour connaître la valeur d'une donnée
 - pour modifier une donnée
 - pour obtenir un certain comportement indépendant des données
 - Application graphique :
 - le **message** que j'envoie à E est « afficher cercle ». Ce message indique que je souhaite visualiser un cercle.
 - Un message que je peux envoyer à C est « taille » ; il doit répondre en donnant sa surface

Méthodes

- ◆ Méthode:

- opération qui permet de répondre à un message
- un message déclenche l'exécution d'une méthode
- Application graphique :
 - *La méthode que E possède pour « afficher » consiste en un ensemble de mécanismes pour l'affichage de la forme à l'écran*
 - *La méthode que C possède pour répondre à « taille » est Surface.*

Messages et Méthodes

- ◆ La manière dont un objet met en œuvre une méthode est cachée de l'appelant
 - Application graphique : lorsqu'un un cercle reçoit le message « taille », il doit répondre en donnant sa surface. Le cercle peut calculer sa surface chaque fois qu'il reçoit ce message, ou il peut la calculer une fois pour toute, et stocker le résultat avec l'aide de l'objet disque et relire ce résultat dans la méthode pour « taille ».
 - Un objet est donc un composant dont ses *internes* sont *cachés* aux autres objets
 - Le but est de pouvoir modifier le comportement d'un objet sans toucher au reste du système !

Messages et Méthodes

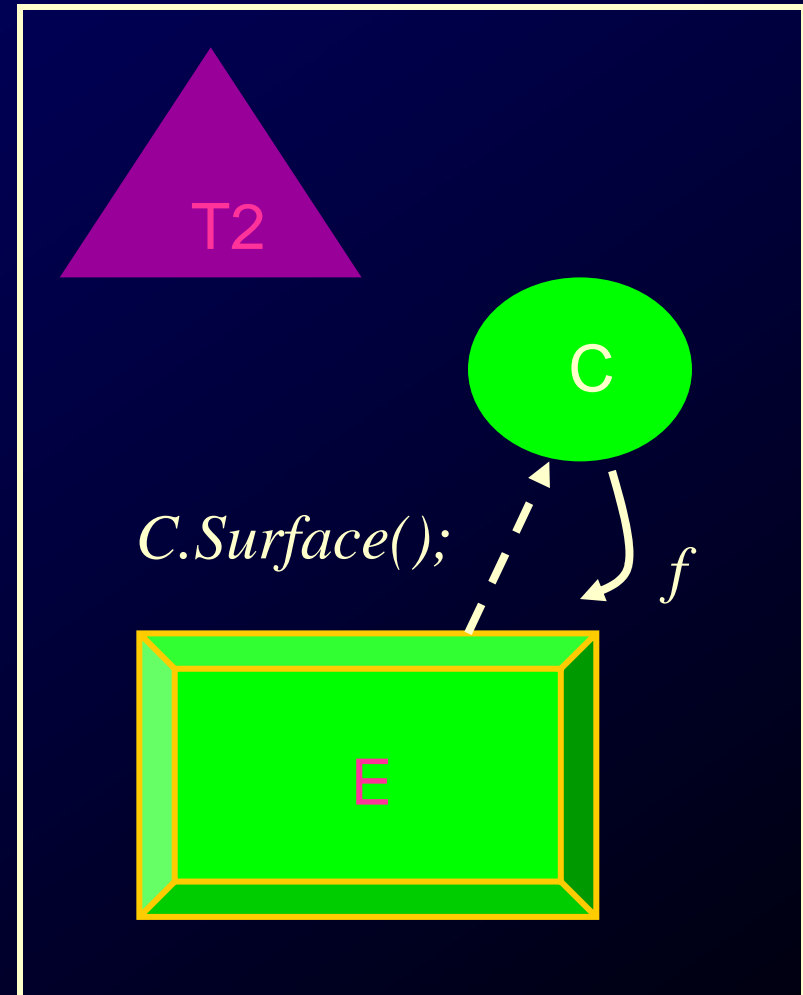
- ◆ Chaque objet possède un *nom unique*
- ◆ L'expéditeur d'un message doit *nommer explicitement* l'objet destinataire
 - Le nom du destinataire n'est connu qu'au moment de l'envoi du message
 - On dit alors qu'il existe une **liaison tardive** entre le message envoyé et la méthode exécutée
 - Application graphique :
 - *Je choisis à l'exécution entre 2 objets triangle.*
 - *Chaque objet est indépendant. La méthode utilisée par un triangle peut être différente de celle de l'autre.*

Messages et Méthodes

Les méthodes sont déclenchées par la réception de *messages*

```
afficherCercle(){  
  C = new Cercle(3,4,5.0);  
  float f = C.Surface();  
  ... }
```

Généralement, un message est un appel de méthodes

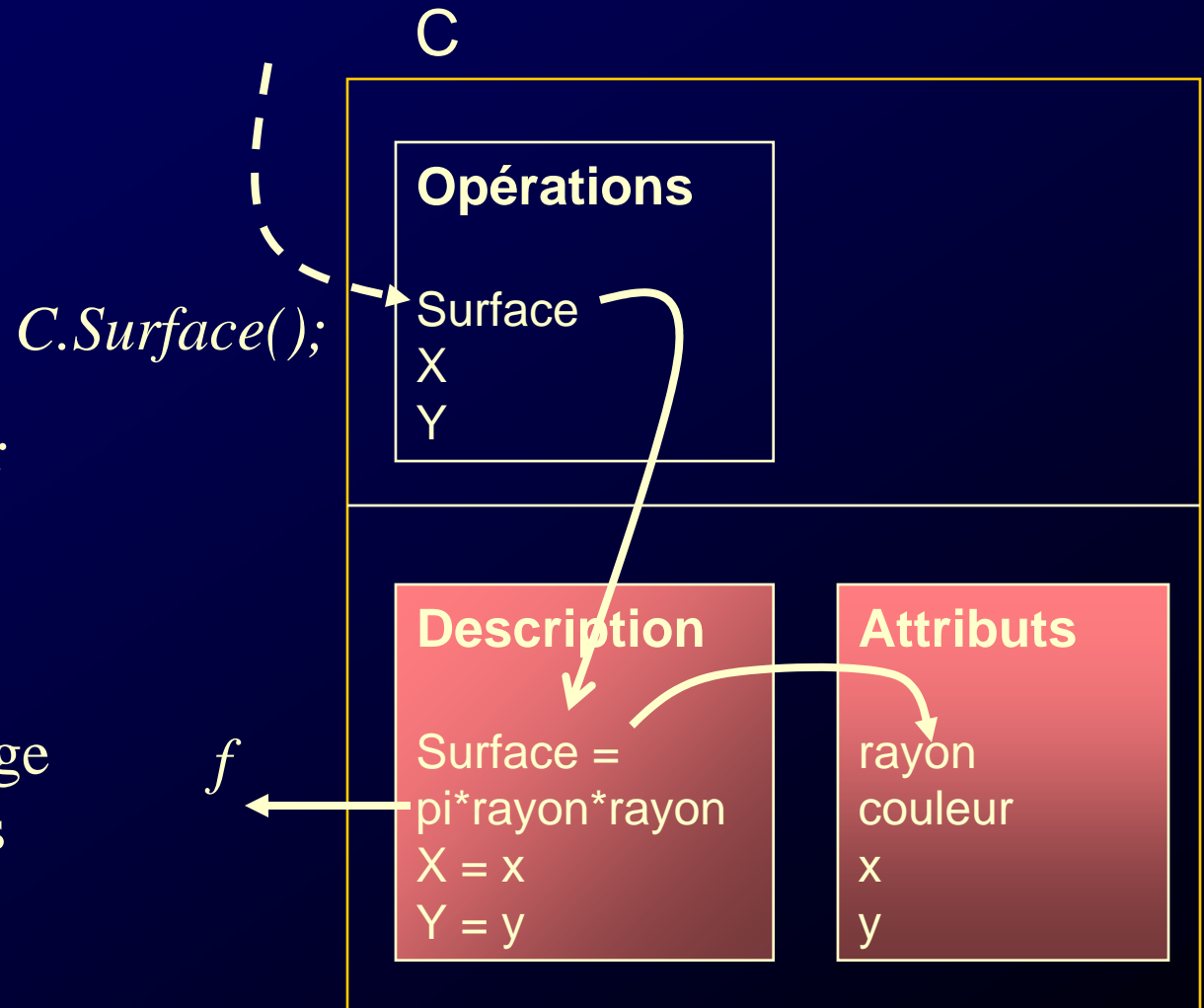


Messages et Méthodes

Les méthodes sont déclenchées par la réception de *messages*

```
afficherCercle(){  
  C = new Cercle(3,4,5.0);  
  float f = C.Surface();  
  ... }  
}
```

Généralement, un message est un appel de méthodes



Appel de méthodes en Java

```
class Ecran {
```

```
    int x, y; float rayon;
```

```
    Cercle C;
```

```
    public Ecran(){
```

```
        C = new Cercle(3,4,5.0);  
    }
```

```
    public afficherCercle() {
```

```
        float f = C.Surface();
```

```
        ...  
    }
```

```
}
```

```
class Cercle {
```

```
    int x, y; float rayon;
```

```
    public Cercle(int a, int b, float r){
```

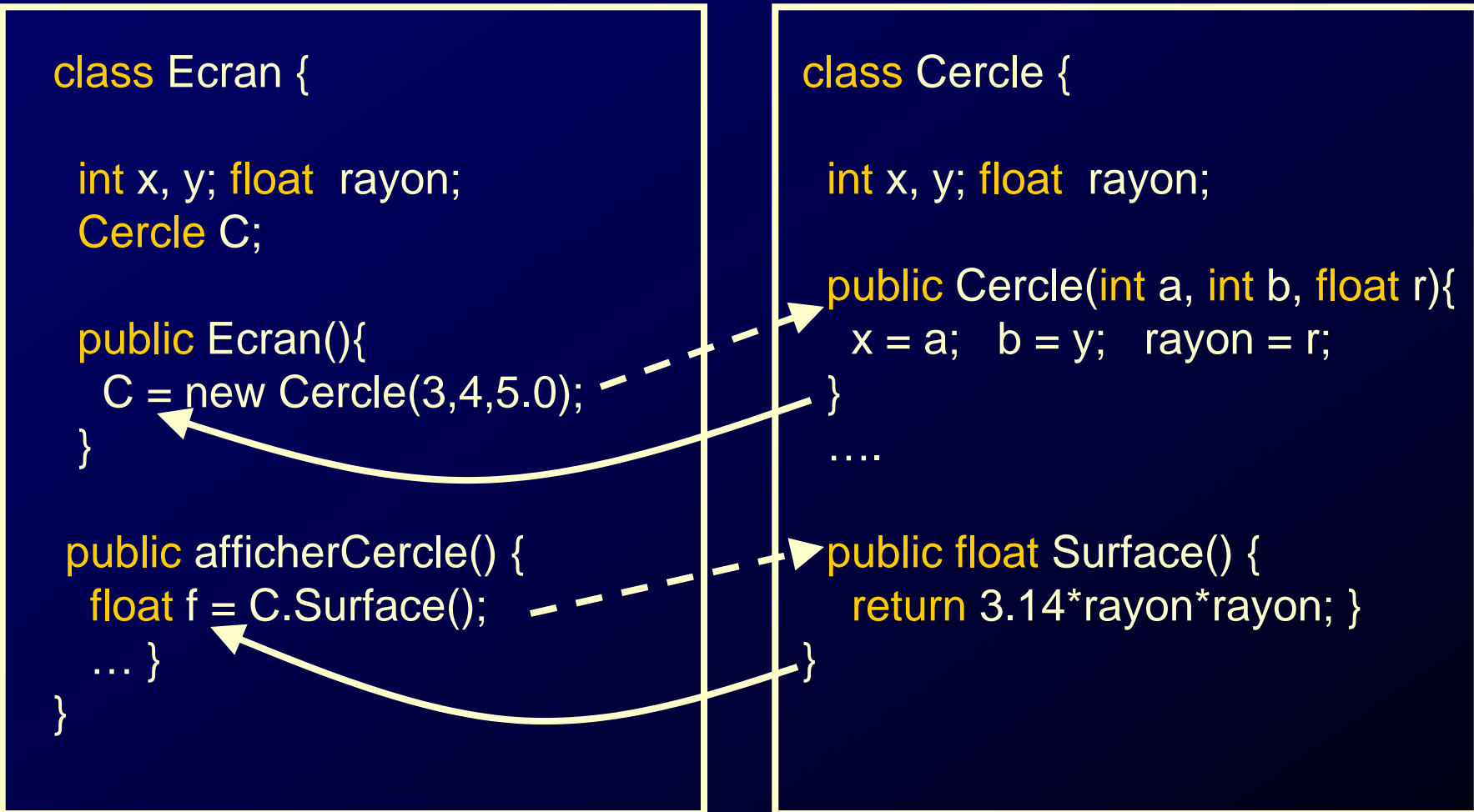
```
        x = a;  b = y;  rayon = r;
```

```
    }
```

```
    ....
```

```
    public float Surface() {
```

```
        return 3.14*rayon*rayon;  
    }
```



Héritage

- ◆ Les classes peuvent parfois avoir des méthodes et attributs en commun
- ◆ Caractéristiques communes à plusieurs classes sont placées dans une classe séparée
 - Les caractéristiques spécifiques à chaque classe sont placées dans des classes particulières
 - La partie commune n'a pas besoin d'être écrite à nouveau, elle est *héritée*

Héritage

♦ Application graphique :

- *On désire des cercles qui en plus de calculer la surface soient capable d'indiquer leur couleur*
- *Le programmeur définit une classe Cercle, et une classe CercleColoré*
- *un **cercle coloré** est aussi un cercle ; il comprend donc les messages « taille », mais en plus « VotreCouleur »*
- *la classe CercleColoré **hérite** de la classe Cercle*

Héritage

- ◆ Lorsqu'une classe B hérite de la classe A
 - Toutes les opérations et attributs définis dans la classe A font partie de la classe B
 - B comprend tous les messages compris par A
 - Application graphique : *La classe CercleColoré comprend le message « taille »*
 - Par défaut, B utilise la même méthode pour tout message hérité
 - Application graphique : *les méthodes Surface() de la classe Cercle et de la classe CercleColoré sont les mêmes. Elles retournent la surface de la forme à l'appelant.*

Héritage

◆ *Nouvelles Méthodes*

- Application graphique : la classe *CercleColoré* peut définir la méthode *getCouleur()* qui retourne la couleur de l'objet

◆ *Over-loading (la sur-charge)*

- Lorsque la classe B hérite de la classe A, elle peut redéfinir ses propres méthodes pour les messages hérités
 - Application graphique : la classe *CercleColoré* peut redéfinir la méthode *Surface()* afin de rendre un résultat avec plus de précisions décimales.

◆ *Héritage et over-loading*

- Les moyens de base pour l'extensibilité de logiciel

Héritage

- ◆ Importance

- Réutilisation
 - Application graphique : à partir de la classe *Cercle*, on peut définir la classe *CercleColoré*, mais aussi *CercleTransparent*, etc.
- Modification à un seul endroit
 - Application graphique : une modification de la méthode *Surface()* dans la classe *Cercle* est immédiatement prise en compte par la classe *CercleColoré*
- Eviter la redondance
 - Pas de duplication, seule les différences sont décrites

Héritage en Java

```
class Cercle {
```

```
    int x, y; float rayon;
```

```
    public Cercle(int a, int b, float r){  
        x = a;  b = y;  rayon = r ;  
    }
```

```
    public int X()  {return x; }  
    public int Y()  {return y; }  
    public float Surface() {  
        return 3.14*rayon*rayon; }  
}
```

```
class CercleColoré extends Cercle{
```

```
    int couleur;
```

```
    public CercleColoré(  
        int a, int b, float r, int c){  
        x = a;  b = y;  rayon = r ;  
        couleur =c;  
    }
```

```
    public int getCouleur(){  
        return couleur; }  
}
```

```
    int x, y; float rayon;  
    public int X()  {return x; }  
    public int Y()  {return y; }  
    public float Surface() {  
        return 3.14*rayon*rayon; }  
}
```

Polymorphisme

- ◆ Au moment de l'envoi d'un message à un objet, on ne connaît pas la classe de l'objet
 - Application graphique :
 - *l'objet E responsable d'afficher à l'écran des formes géométriques manipule indifféremment des cercles, des carrés, ou des triangles*
- ◆ Destinataire du message qui décide de l'effet et pas l'expéditeur
- ◆ Polymorphisme restreint est lié à l'héritage
 - *l'objet E manipule des objets de la classe Cercle, sans savoir s'il s'agit en fait d'un cercle ou d'un cercle coloré*