

COURS JAVA UTM TINDANO

ref:H Bersini

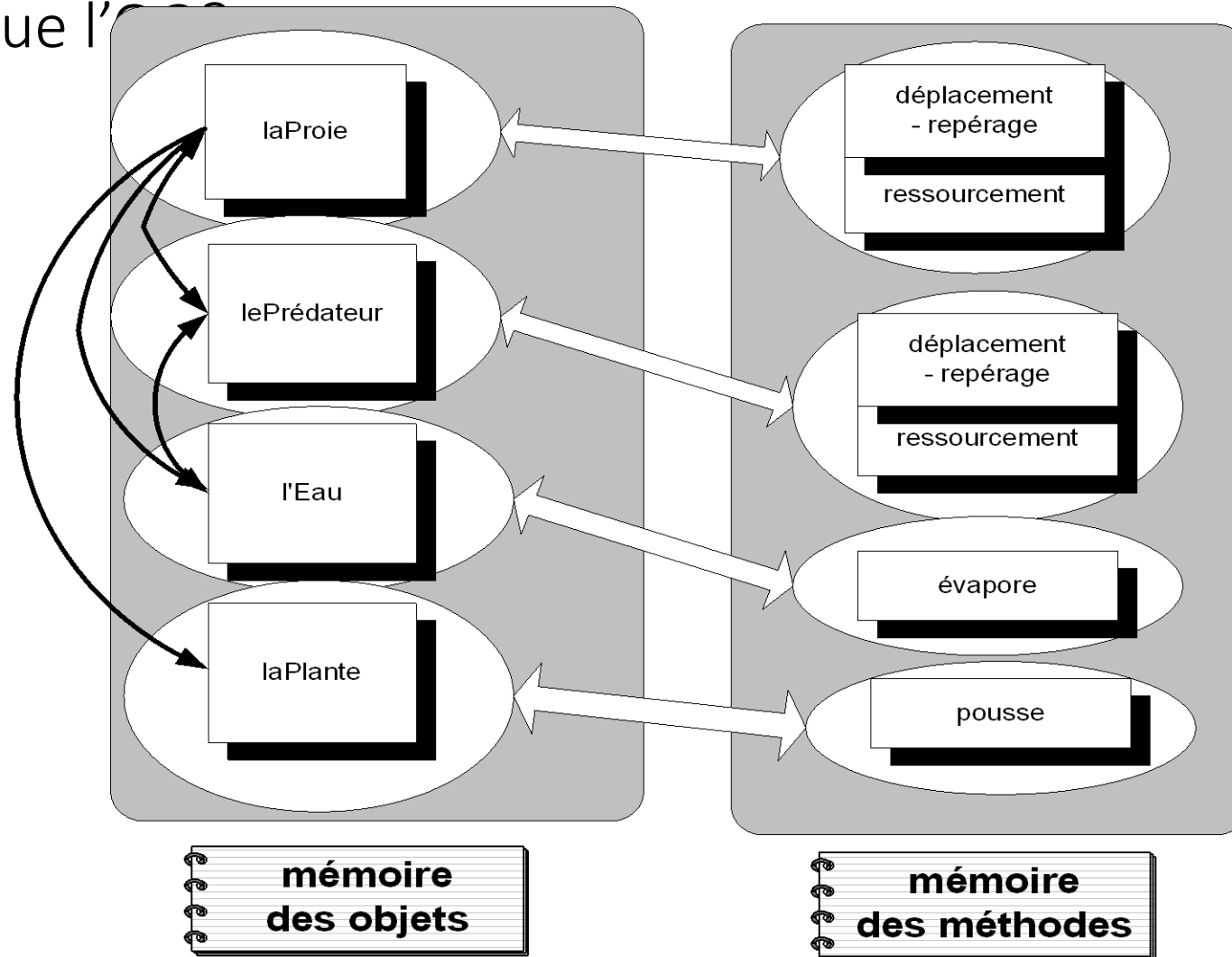
Les concepts de l'OO

Qu'est-ce que l'OO?

- Comment procéder?
 - Identifier les acteurs du problème: Proie/Prédateur/Plante/Eau
 - Identifier les actions ou comportements de chaque acteur
 - Le prédateur:
 - se déplace en fonction des cibles, peut boire l'eau et manger la proie
 - La proie:
 - se déplace en fonction des cibles, peut boire l'eau et manger la plante
 - La plante:
 - pousse et peut diminuer sa quantité
 - L'eau:
 - s'évapore et peut diminuer sa quantité

Les concepts de l'OO

Qu'est-ce que l'OO ?



Les concepts de l'OO

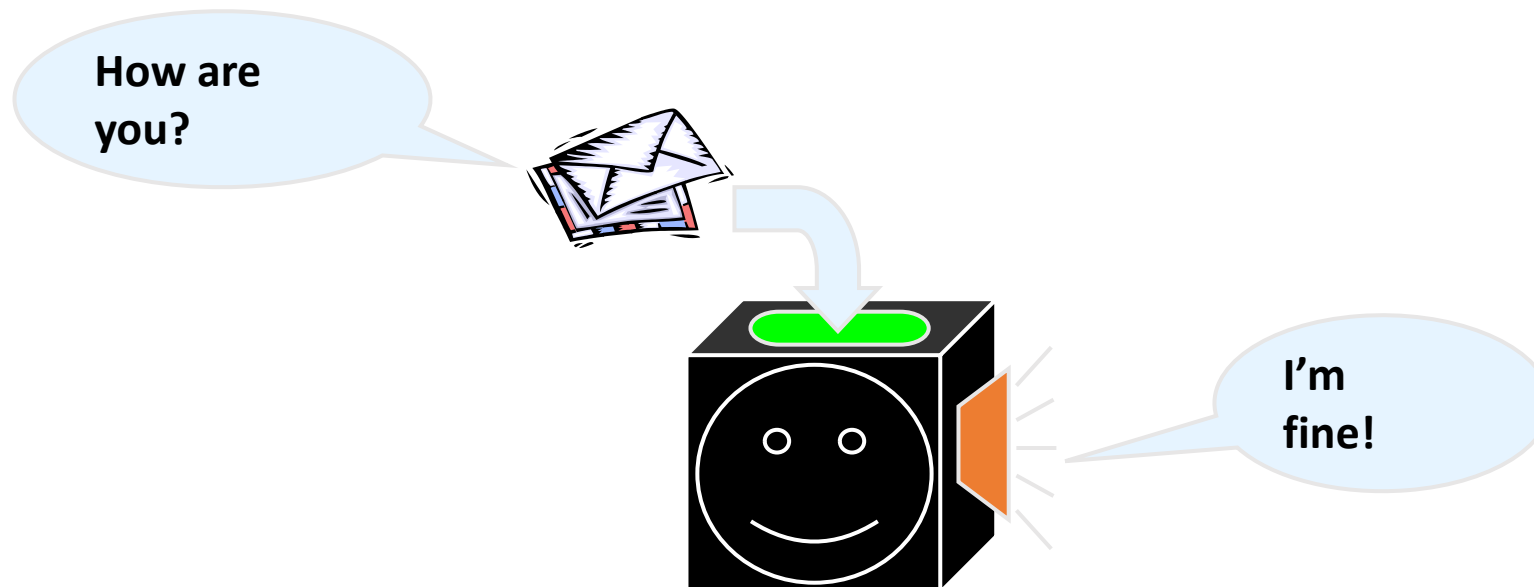
Qu'est-ce que l'OO?

- La programmation consistera donc à définir les différents acteurs (objets), leurs caractéristiques et comportements et à les faire interagir.
- La programmation OO permet donc de découper les problèmes informatiques non en termes de fonctions (que se passe-t-il?) mais en termes de données (qui intervient dans le problème?) et des services que celles-ci peuvent rendre au reste du programme (les comportements).
- Programmer en OO, c'est donc définir des objets et les faire interagir.

Les concepts de l'OO

Qu'est-ce qu'un objet?

- La programmation OO consiste à définir des objets et à les faire interagir
- Qu'est-ce qu'un objet?
 - ➔ Une boîte noire qui reçoit et envoie des messages



Les concepts de l'OO

Qu'est-ce qu'un objet?

- Du code → Traitements composés d'instructions
→ Comportements ou **Méthodes**
- Des données → L'information traitée par les instructions et qui caractérise l'objet → Etats ou **Attributs**
- Données et traitements sont donc indissociables
- Les traitements sont conçus pour une boîte en particulier et ne peuvent pas s'appliquer à d'autres boîtes

Les concepts de l'OO

Qu'est-ce qu'un objet?

- Quelques exemples?

Objet	Attributs	Méthodes
Chien	Nom, race, âge, couleur	Aboier, chercher le baton, mordre, faire le beau
Téléphone	N° , marque, sonnerie, répertoire, opérateur	Appeler, Prendre un appel, Envoyer SMS, Charger
Voiture	Plaque, marque, couleur, vitesse	Tourner, accélérer, s'arrêter, faire le plein, klaxonner

Exercices

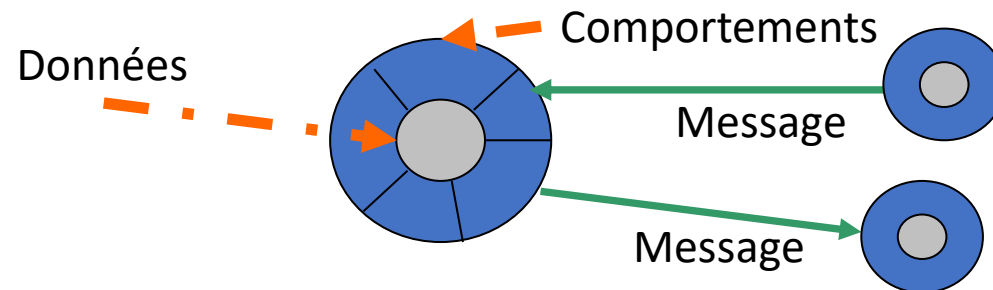
- Comptes en banque
 - Quelles sont les variables d'états caractéristiques d'un compte en banque?
 - Devraient-elles être accessibles ou non de l'extérieur?
 - Quels sont les comportements possibles d'un compte en banque?
 - Devraient-elles être accessibles ou non de l'extérieur?

L'encapsulation

- Pourquoi une boîte noire?

- ➔ L'utilisateur d'un objet ne devrait jamais avoir à plonger à l'intérieur de la boîte
- ➔ Toute l'utilisation et l'interaction avec l'objet s'opère par messages
- ➔ Les messages définissent l'interface de l'objet donc la façon d'interagir avec eux
- ➔ Il faut et il suffit de connaître l'interface des messages pour pouvoir exploiter l'objet à 100%, sans jamais avoir à connaître le contenu exact de la boîte ni les traitements qui l'animent
- ➔ L'intégrité de la boîte et la sécurité de ses données peut être assurée
- ➔ Les utilisateurs d'un objet ne sont pas menacés si le concepteur de l'objet en change les détails ou la mécanique interne

➔ **ENCAPSULATION**



La classe

Un objet sans classe n'a pas de classe

- Comment les objets sont-ils définis?

- ➔ Par leur classe, qui détermine toutes leurs caractéristiques
 - ➔ Nature des attributs et comportements possibles
- ➔ La classe détermine tout ce que peut contenir un objet et tout ce qu'on peut faire de cet objet
- ➔ Classe = Moule, Définition, ou Structure d'un objet
- ➔ Objet = Instance d'une classe
- ➔ Une classe peut-être composite \Leftrightarrow peut contenir elle-même des objets d'autres classes
 - ➔ Ex: Une voiture contient un moteur qui contient des cylindres...
 - ➔ Ex: Un chien possède des pattes...

La classe

Un objet sans classe n'a pas de classe

- Quelques exemples?

- La classe « chien » définit:

- Les attributs d'un chien (nom, race, couleur, âge...)
 - Les comportements d'un chien (Aboier, chercher le bâton, mordre...)

- Il peut exister dans le monde plusieurs objets (ou instances) de chien

Classe	Objets
Chien	Mon chien: Bill, Teckel, Brun, 1 an Le chien de mon voisin: Hector, Labrador, Noir, 3 ans
Compte	Mon compte à vue: N° 210-1234567-89, Courant, 1.734 €, 1250 € Mon compte épargne: N° 083-9876543-21, Epargne, 27.000 €, 0 €
Voiture	Ma voiture: ABC-123, VW Polo, grise, 0 km/h La voiture que je viens de croiser: ZYX-987, Porsche, noire, 170 km/h

La classe Un objet sans classe n'a pas de classe

Déclaration
de la classe

Variables d'instance
ou « champs »

Définition du
constructeur

Méthodes d'accès

Définition
des
méthodes

```
import java.lang.*;
private String marque;
public class Voiture {
    private int vitesse;
    private double reserveEssence;
    public Voiture(String m,int v,double e){
        marque=m ;
        vitesse=v;
        reserveEssence=e;
    }
    public String getMarque(){return marque;}
    public void setMarque(String m){marque=m;}
    ...
    public void accelere(int acceleration) {
        vitesse += acceleration;
    }
    public void ravitaille(double quantite){
        reserveEssence+=quantite;
    }
}
```

La classe

Déclaration des attributs

- Les attributs d'un objet sont définis par ses variables membres
- Pour définir les attributs d'un objet, il faut donc définir les variables membres de sa classe
- Les principes suivants gouvernent toutes les variables en Java:
 - Une variable est un endroit de la mémoire à laquelle on a donné un nom de sorte que l'on puisse y faire facilement référence dans le programme
 - Une variable a une valeur, correspondant à un certain type
 - La valeur d'une variable peut changer au cours de l'exécution du programme
 - Une variable Java est conçue pour un type particulier de donnée
 - Une déclaration typique de variable en Java a la forme suivante:
 - `int unNombre;`
 - `String uneChaineDeCaracteres;`

La classe

Déclaration des attributs

- Rappel: toute variable doit être déclarée et initialisée
- Les variables membres sont des variables déclarées à l'intérieur du corps de la classe mais à l'extérieur d'une méthode particulière, c'est ce qui les rend accessibles depuis n'importe où dans la classe.
- La signature de la variable :
 - Les modificateurs d'accès: indiquent le niveau d'accessibilité de la variable
 - [static]: permet la déclaration d'une variable de classe
 - [final]: empêche la modification de la variable (→ Crée une constante)
 - [transient]: on ne tient pas compte de la variable en sérialisant l'objet
 - [volatile]: pour le multithreading
 - Le type de la variable (ex: int, String, double, RacingBike,...)
 - Le nom de la variable (identificateur)
- Exemples:
 - `private int maVitesse; // Je possède un entier
« maVitesse »`
 - `private final String maPlaque; // « maPlaque » est
constante`

optionnels



La classe

Déclaration des attributs

Les modificateurs d'accès qui caractérisent l'encapsulation sont justifiées par différents éléments:

- Préservation de la sécurité des données
 - Les données privées sont simplement inaccessibles de l'extérieur
 - Elles ne peuvent donc être lues ou modifiées que par les méthodes d'accès rendues publiques
- Préservation de l'intégrité des données
 - La modification directe de la valeur d'une variable privée étant impossible, seule la modification à travers des méthodes spécifiquement conçues est possible, ce qui permet de mettre en place des mécanismes de vérification et de validation des valeurs de la variable
- Cohérence des systèmes développés en équipes
 - Les développeurs de classes extérieures ne font appel qu'aux méthodes et, pour ce faire, n'ont besoin que de connaître la signature. Leur code est donc indépendant de l'implémentation des méthodes

La classe

Déclaration des attributs

- En java, les modificateurs d'accès sont utilisés pour protéger l'accessibilité des attributs et des méthodes.
- Les accès sont contrôlés en respectant le tableau suivant:

Mot-clé	classe	package	sous classe	world
<code>private</code>	Y			
<code>protected</code>	Y	Y	Y	
<code>public</code>	Y	Y	Y	Y
<code>[aucun]</code>	Y	Y		

Seul les membres publics sont visibles depuis le monde extérieur.

Une classe a toujours accès à ses membres.

Les classes d'un même package protègent uniquement leurs membres privés (à l'intérieur du package)

Une classe fille (ou dérivée) n'a accès qu'aux membres publics et protected de la classe mère.

Bien utiliser les accesseurs et comprendre l'encapsulation

- De manière générale, on évitera de déclarer un attribut “public”
- Ou alors, n'importe qui pourra modifier la variable.
- Il faudra paramétrer les accès en utilisant les packages : seules les classes du même package (que vous contrôlez vous même!!!) pourront modifier les variables packagées.
- Pour packager une variable, une méthode ou une classe. ne mettez aucun accesseur devant celle-ci. Créez juste un package (càd. un répertoire dans le système de fichiers) et déclarez que votre classe appartient au package au tout début du fichier via :

```
package nomdepackage;
```

Exercices

- Création d'une classe compte en banque packagée
 - Créez un nouveau projet Eclipse
 - Créez un nouveau package « banksys »
 - Créez-y une nouvelle classe « CompteEnBanque »
 - Déclarez les attributs de la classe
- Orienté objet et bases de données
 - Quelle analogie peut-on établir entre orienté objet et bases de données?
 - Quelles sont les limites de cette analogie?

Exercices

• Similitudes avec les bases de données?

- Classe → Table
- Attribut → Champ / Colonne
- Objet → Tuple (enregistrement) de la table
- Valeur → Valeur du champ ou de la colonne

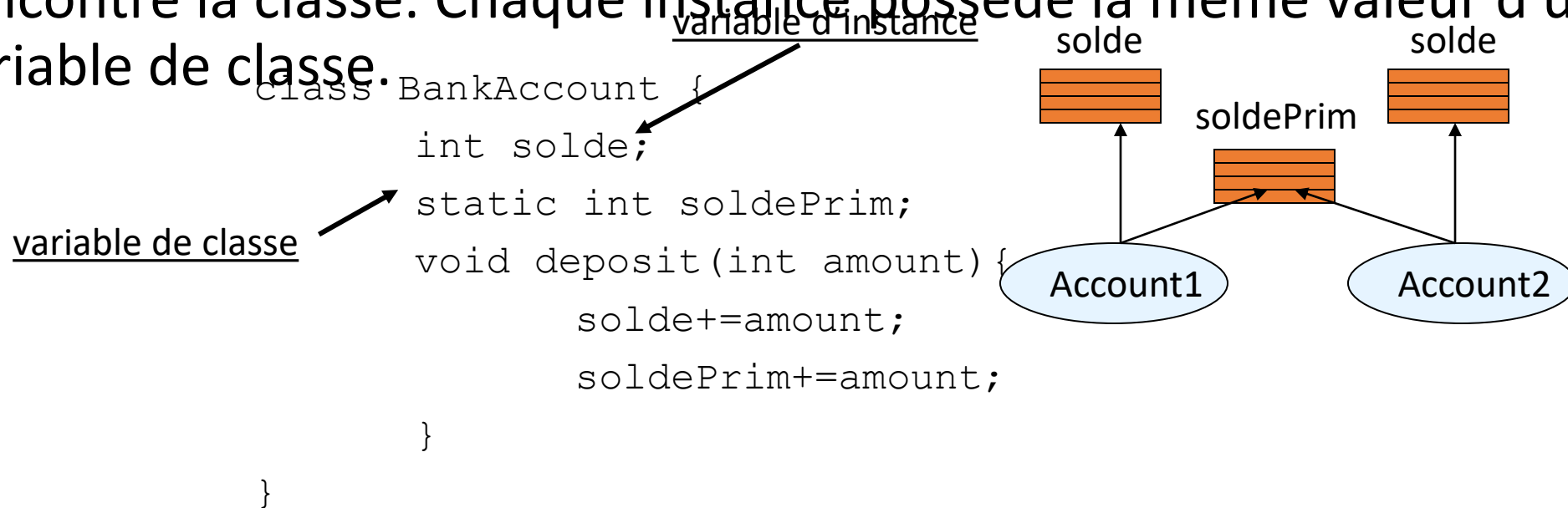
Marque	Modele	Serie	Numero
Renault	18	RL	4698 SJ 45
Renault	Kangoo	RL	4568 HD 16
Renault	Kangoo	RL	6576 VE 38
Peugeot	106	KID	7845 ZS 83
Peugeot	309	chorus	7647 ABY 82
Ford	Escort	Match	8562 EV 23

- Mais il y a des limites à cette analogie (qui apparaîtront plus tard)

La classe

Déclaration des attributs

- Chaque objet a sa propre “mémoire” de ses variables d’instance
- Le système alloue de la mémoire aux variables de classe dès qu’il rencontre la classe. Chaque instance possède la même valeur d’une variable de classe.



La classe

Déclaration des attributs

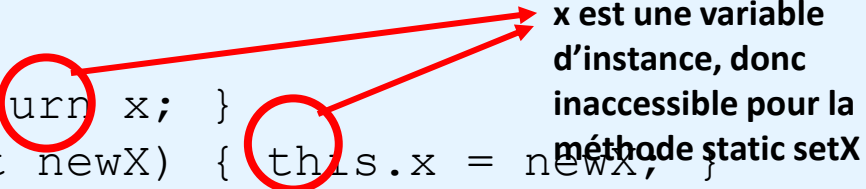
- Variables et méthodes statiques

- Initialisées dès que la classe est chargée en mémoire
- Pas besoin de créer un objet (instance de classe)

- Méthodes statiques

- Fournissent une fonctionnalité à une classe entière
- Cas des méthodes non destinées à accomplir une action sur un objet individuel de la classe
- Exemples: `Math.random()`, `Integer.parseInt(String s)`, `main(String[] args)`
- Les méthodes statiques ne peuvent pas accéder aux variables

```
class AnIntegerNamedX {  
    int x;  
    static public int x() { return x; }  
    static public void setX(int newX) { this.x = newX; }  
}
```



x est une variable d'instance, donc inaccessible pour la méthode static setX

La classe

Le constructeur on l'a vu, les variables doivent être initialisées

- Or la déclaration des attributs n'inclut généralement pas d'initialisation
- De plus, comment pourrait-on connaître à l'avance la valeur des différentes variables membres, puisqu'elle sera propre à chaque instance (objet) de la classe? (Chaque voiture a sa propre marque, sa propre couleur et sa propre vitesse)
- Il faut donc définir dans la classe un mécanisme permettant de donner une valeur aux variables membres (aux états) de l'objet
- Ce mécanisme s'appelle le « constructeur »
- Une classe doit toujours avoir au minimum un constructeur
- Le constructeur recevra les valeurs à attribuer aux variables et les y affectera
- Le constructeur servira ainsi à « matérialiser » la classe et sera donc appelé chaque fois qu'un objet de la classe doit être créé
- Le constructeur est donc la « Recette » pour fabriquer un objet

La classe

Le constructeur

- A le même nom que la classe
- Utilise comme arguments des variables initialisant son état interne
- On peut surcharger les constructeurs, i.e définir de multiples constructeurs (plusieurs recettes existent pour fabriquer un même objet avec des ingrédients différents)
- Il existe toujours un constructeur. S'il n'est pas explicitement défini, il sera un constructeur par défaut, sans arguments
- Signature d'un constructeur:
 - Modificateur d'accès (en général public)
 - Pas de type de retour
 - Le même nom que la classe
 - Les arguments sont utilisés pour initialiser les variables de la classe

```
public Voiture(String m,int
    v,double e)
{
    marque=m ;
    vitesse=v;
    reserveEssence=e;
}

public Voiture(String m)
{
    marque=m ;
    vitesse=0;
    reserveEssence=0;
}
```

Exercices

- Comptes en banque
 - Créer deux constructeurs différents dans la classe CompteEnBanque

Interaction entre objets

Création d'objets

- Notons toutefois qu'il ne « se passe » toujours rien dans notre programme à ce stade
- En effet, nous avons conçu deux classes, dont nous avons défini les caractéristiques, les constructeurs et les comportements possibles, mais nous n'avons toujours créé aucun objet
- Nous n'avons en fait encore que défini ce que seraient un compte en banque ou une personne si nous en créions dans notre programme
- Pour qu'il « se passe » quelque chose, il faudrait donc que nous matérialisions nos classes, autrement dit que nous lesinstancions, ce qui signifie que nous « créions des objets »
- Pour ce faire, il faut appeler le constructeur de la classe dont nous souhaitons créer une instance

Interaction entre objets

Création d'objets

- L'appel au constructeur
 - Se fait pour initialiser un objet
 - ➔ Provoque la création réelle de l'objet en mémoire
 - ➔ Par l'initialisation de ses variables internes propres
- Se fait par l'emploi du mot clé « **new** »

```
Voiture v1, v2;
```

```
v1 = new Voiture("Peugeot", 120, 35.3);
```

```
v2 = new Voiture("Ferrari");
```

Interaction entre objets

Création d'objets

Déclaration et création d'objets

- **Déclaration** : `Point p;`

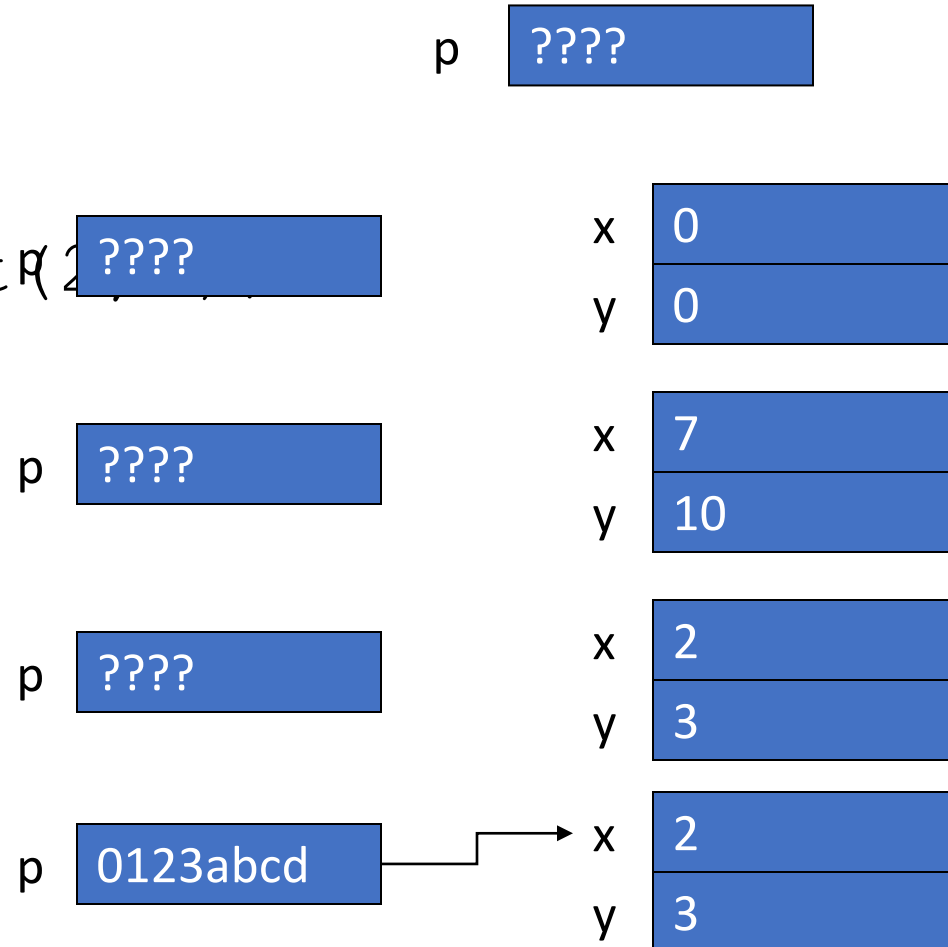
- **Création** : `p = new Point(2, 3);`

1. CHERCHER UNE PLACE

2. Assignation d'une valeur

3. Exécution du constructeur

4. Création du pointeur



Exercices

- Comptes en banque
 - Création d'une classe Principale
 - Créer une classe « Principale » représentant un scénario possible d'utilisation de vos classes.
 - Mettez cette classe en dehors du package.
 - Y définir une méthode « main », dans laquelle déclarer quelques objets de type « CompteEnBanque »
 - Instancier arbitrairement ces quelques comptes
 - Réaliser quelques opérations sur les comptes (retraits, dépôts, virements, calcul d'intérêts, etc.)
 - Afficher tous les comptes à l'écran
 - Jouez avec les accesseurs et constatez les conséquences au niveau de la méthode main.

La classe

Déclaration des méthodes

- Quid des comportements?
 - La classe définit aussi les comportements des objets
 - Les comportements sont les messages que les objets de la classe peuvent comprendre ➔ « Aboie », « Va chercher », « Fais le beau »
 - Les traitements ou instructions à réaliser lorsqu'un message en particulier est envoyé à un objet sont définis dans la classe mais restent cachés (cf. boîte noire)
 - Certains messages requièrent des renseignements complémentaires ➔ « Va chercher... le bâton », « Aboie... 3 fois très fort »
 - Une méthode peut renvoyer une valeur ou un objet à son expéditeur

La classe

Déclaration des méthodes

- Quid des comportements?
 - Pour déclencher un traitement, il faut donc envoyer un message à l'objet concerné → On n'invoque jamais une fonction dans le vide
 - L'opérateur d'envoi de messages est le point « . »
 - Pour envoyer un message, il faut toujours préciser à quel objet en particulier on le destine
 - Ex: **monChien.vaChercher(leBaton)**
 - Si on ne précise pas le destinataire, Java considère qu'on est son propre destinataire
 - Ex: `vaChercher(leBaton)`
 - C'est celui qui envoie l'ordre qui le reçoit
 - Ce qui peut aussi s'écrire: `this.vaChercher(leBaton)`
- « this » signifie en effet « l'objet présent »

La classe

Déclaration des méthodes

- Une méthode est composée de

```
public void deposit (int amount) {  
    solde+=amount ;  
}
```

Sa déclaration

Son corps

- Signature d'une méthode:

Signature

- Modificateurs d'accès : public, protected, private, aucun
- [modificateurs optionnels] : static, native, synchronized, final, abstract
- Type de retour : type de la valeur retournée
- Nom de la méthode (identificateur)
- Listes de paramètres entre parenthèses (peut être vide mais les parenthèses sont indispensables)
- [exception] (throws Exception)

- Au minimum:

- La méthode possède un identificateur et un type de retour
- Si la méthode ne renvoie rien → le type de retour est void

- Les paramètres d'une méthode fournissent une information depuis l'extérieur du "scope" de la méthode (idem que pour le constructeur)

La classe

Déclaration des méthodes

- Qu'est-ce que « l'interface » d'une méthode
 - L'interface d'une méthode, c'est sa signature
 - Cette signature, qui définit l'interface de la méthode, correspond en fait au message échangé quand la méthode est appelée
 - Le message se limite de fait uniquement à la signature de la méthode
 - Type de retour
 - Nom
 - Arguments
 - L'expéditeur du message n'a donc jamais besoin de connaître l'implémentation ou corps de la méthode
- On a donc:
 - Déclaration = Signature = Message de la méthode
 - Bloc d'instruction = Corps = Implémentation de la méthode

Exercices

- Comptes en banque
 - Déclarer et implémenter les méthodes (comportements) de la classe compte en banque
 - Définissez en particulier une méthode « afficheToi » qui affiche les caractéristiques du comptes à l'écran
 - Utilisez pour cela la méthode
`System.out.println (« ... ») ;`

Interaction entre objets

Qu'est-ce qu'un envoi de message?

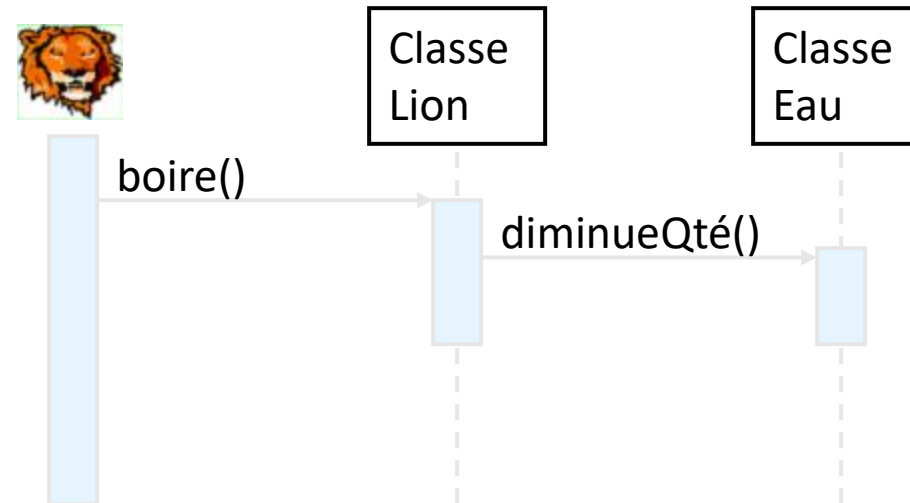
- Tout ce qui se produit dans un programme OO est le résultat d'objets qui interagissent
- L'interaction entre deux objets consiste en l'envoi d'un message de l'objet A à l'objet B
- Cet envoi de message est comme un ordre ou une question qu'un objet adresse à l'autre
- Techniquement, un envoi de message se matérialise par l'appel d'une méthode définie dans la classe de l'objet destinataire
- Exemple:
 - Soit un objet Maître et un objet Chien
 - Le maître peut envoyer un message au chien:
➔ `monChien.faisLeBeau()` ;

Interaction entre objets

Qu'est-ce qu'un envoi de message?

• Exemple?

- Quand le lion s'abreuve, il provoque une diminution de la quantité d'eau
- Ce n'est pas le lion qui réduit cette quantité
- Le lion peut seulement envoyer un message à l'eau, lui indiquant la quantité d'eau qu'il a consommée
- L'eau gère seule sa quantité, le lion gère seul son énergie et ses déplacements



Interaction entre objets

Association

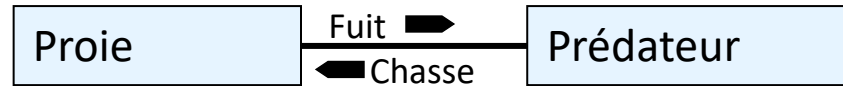
• Comment le lion connaît-il le message à envoyer à l'eau?

- Pour pouvoir envoyer un message à l'eau, il faut que le lion connaisse précisément l'interface de l'eau
- La classe lion doit pour cela « connaître » la classe eau
- Cette connaissance s'obtient par l'établissement d'une communication entre les deux classes
- De tels liens peuvent être
 - Circonstanciels et passagers → **Dépendance**
(le lien ne se matérialise que dans une méthode)
 - Persistants → **Association**
- On distingue 3 types de liens d'association, du + faible au + fort:
 - Association simple (il y a envoi de messages entre les classes)
 - Agrégation faible (une classe possède un attribut de l'autre type)
 - Agrégation forte = Composition (l'agrégé n'existe pas sans son contenant)

Interaction entre objets

Association

- L'association entre classes



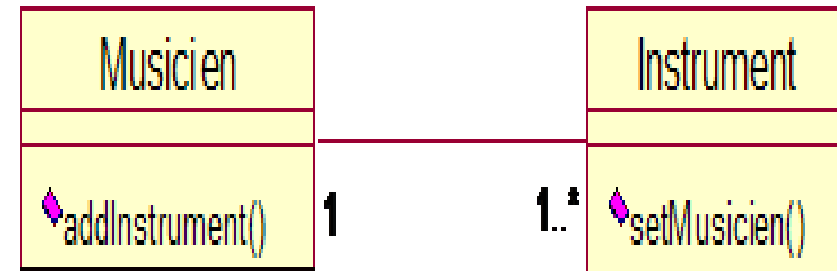
- Multiplicité des associations



Interaction entre objets

Association

```
class Musicien {  
    private List<Instrument> mesInstruments = new  
        List<Instrument>();  
    public Musicien() {}  
    public void addInstrument(Instrument unInstrument) {  
        mesInstruments.add(unInstrument);  
        unInstrument.setMusicien(this);  
    }  
}  
  
class Instrument {  
    private Musicien monMusicien;  
    public Instrument() {}  
    public void setMusicien(Musicien monMusicien) {  
        this.monMusicien = monMusicien;  
    }  
}
```



Interaction entre objets

Association

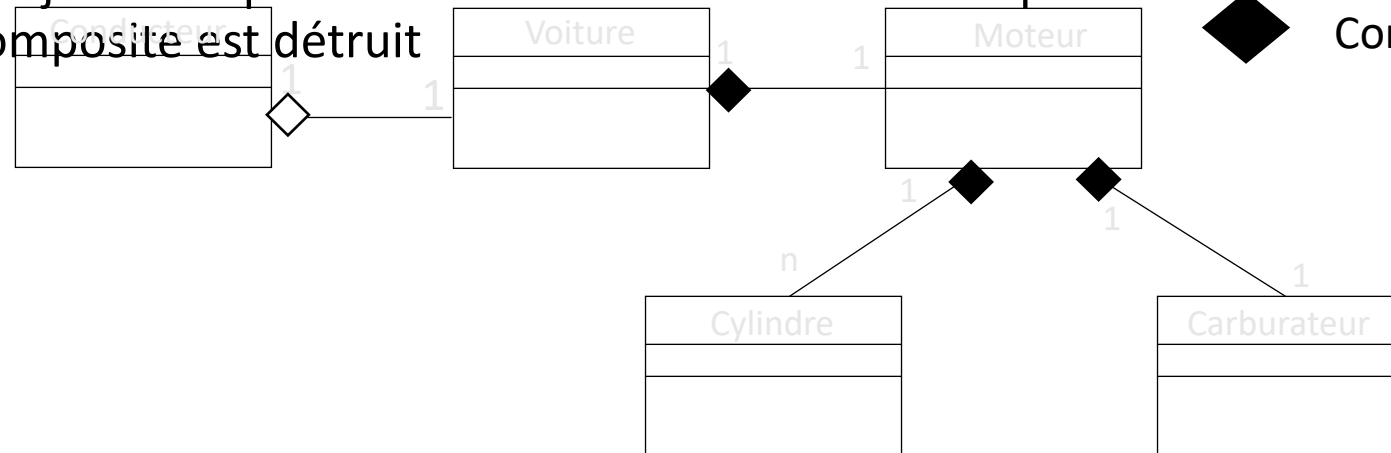
- Deux types d'association particuliers:

- L'agrégation faible

- L'objet « agrégeant » contient une ou plusieurs instances de la classe « agrégée »

- L'agrégation forte ou Composition

- L'objet « composé » est créé dans la classe « composite » et est détruit si la l'objet composite est détruit



◊ Agrégation faible
◆ Composition

Interaction entre objets

Association

- Une classe peut être associée à elle-même: Auto-association



- L'association peut être directionnelle (les messages ne sont envoyés que d'une classe vers l'autre)



Modéliser les classes avec des diagrammes UML

Diagramme de classes

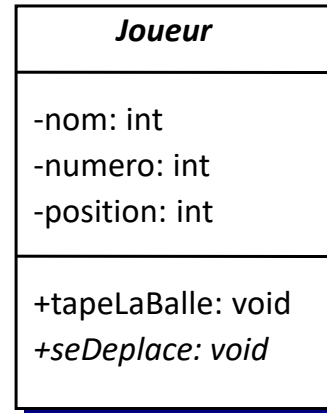
- Le but du diagramme de classes est de représenter les classes au sein d'un modèle
- Dans une application OO, les classes possèdent:
 - Des attributs (variables membres)
 - Des méthodes (fonctions membres)
 - Des relations avec d'autres classes
- C'est tout cela que le diagramme de classes représente
- L'encapsulation est représentée par:
 - (private), + (public), # (protected)
- Les attributs s'écrivent:
 - /+/# nomVariable : Type
- Les méthodes s'écrivent:
 - /+/# nomMethode(Type des arguments) : Type du retour ou « *void* »

Nom Classe
Attributs
Méthodes()

Modéliser les classes avec des diagrammes UML

Diagramme de classes

- Exemple



Modéliser les classes avec des diagrammes UML

Représenter les relations entre classes


- Les relations d'héritage sont représentées par:

- A  B signifie que la classe A hérite de la classe B


- L'association est représentée par:

- A  B signifie que la classe A envoie des messages à la classe B grâce à ses attributs de type B

- L'agrégation faible est représentée par:

- A  B signifie que la classe A a pour caractéristique un ou plusieurs attributs de type B

- L'agrégation forte (ou composition) est représentée par:

- A  B signifie que les objets de la classe B ne peuvent exister qu'au sein d'objets de type A où ils sont créés

Exercices

Comptes en banque

- Créer une classe « Personne » avec quelques attributs, méthodes et constructeurs
- Instancier la classe Personne en tant qu'attribut de la classe CompteEnBanque
- Faites en sorte que le CompteEnBanque, en s'affichant à l'écran, indique le nom de son titulaire
- Une personne peut posséder plusieurs compte en banque
- **N'oubliez pas Faites d'abord un diagramme de classes!**
- Définissez en particulier les 3 méthodes suivantes dans la classe Personne:
 - Une méthode « getSoldeTotal(): double » qui renverra le solde cumulé de tous les comptes de la personne
 - Une méthode « afficheToi() » qui affichera à l'écran les caractéristiques de la personne dont le nombre de compte qu'elle possède ainsi que le solde total de ses comptes et puis listera tous ses comptes
 - Ajouter une méthode « ouvrirCompte(...): void » dans la classe Personne qui attend un compte en banque en paramètre et le stocke dans son tableau

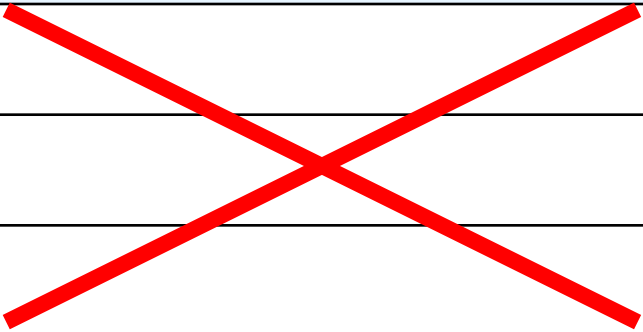



Interaction entre objets

Association

- L'association se matérialise donc par la déclaration d'objets d'un certain type en tant que variables membres d'une autre classe
- Ce faisant, il en résulte un nouveau type de variable, propre aux langages OO, et différent des variables « primitives » que l'on manipule traditionnellement en informatique
- De fait, au lieu de simples variables numérique, booléennes ou textuelles, nous pouvons aussi bien déclarer et manipuler des variables « objet » comme un chien, une voiture, un client, ou encore un compte en banque
- On appelle ces variables « variables de référence » par opposition aux « variables primitives »
- Leur caractéristique principale est qu'elles n'ont pas pour valeur l'objet qu'elle désigne, mais seulement la référence de celui-ci, c'est-à-dire l'adresse à laquelle l'objet qu'elles désignent se trouve dans la mémoire du programme → On parle de « pointeurs » (implicites)

Interaction entre objets

Variables de référence

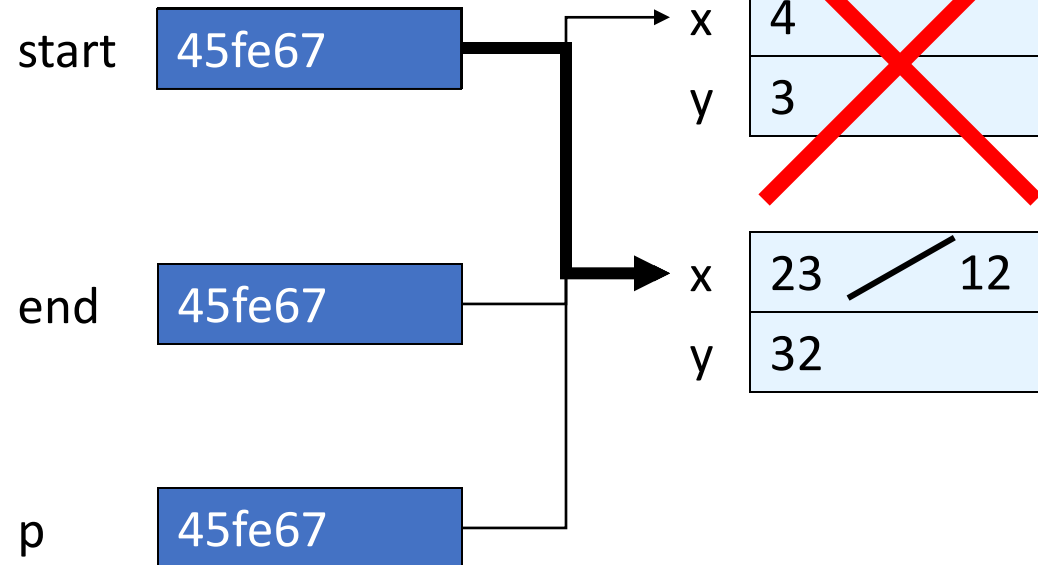
Mémoire des référents	Mémoire des valeurs	Mémoire des objets
int unNombre	10	
double unNombreDecimal	27.33	
boolean unBooléen	<i>true</i>	
Chien monChien	ab123d	
Chien leChienQueTuAsVu	ab123d	
Voiture uneVoitureDeCourse	e4f5a6	
Voiture maVoiture	d12e45	

Interaction entre objets

Variables de référence

Assignation d'un type de référence

- `Point start=new Point(4,3);`
- `Point end=new Point(23,32);`
- `Point p=end;`
- `p.x=12;`
- `start=p;`



Il n'y a désormais plus de référence vers l'ancien Point « start », il sera donc détruit par le Garbage Collector

Exercices

- Analyser la classe Point
 - Que se passe-t-il à l'exécution du programme?
 - Exécuter le programme pour vérifier

Interaction entre objets

Variables de référence

- Java est un langage dit « fortement typé »:
 - Toute variable doit être déclarée
 - Dès sa déclaration, une variable se voit attribuer un type donné
 - Une variable ne peut jamais changer de type
 - Le type de données précise
 - les valeurs que la variable peut contenir ou le type d'objet qu'elle peut désigner (une variable de type « Chat » ne peut pas désigner un objet de type « Chien »)
 - les opérations que l'on peut réaliser dessus
 - ➔ S'il s'agit d'une variable primitive, uniquement les opérations arithmétiques ou binaires correspondantes
 - ➔ S'il s'agit d'une variable de référence, uniquement les méthodes définies dans la classe correspondant au type de la variable

Interaction entre objets

Qu'est-ce qu'un envoi de message?

- Les arguments d'une méthode peuvent être de deux types
 - Variable de type primitif
 - Objet
- Lorsque l'argument est une variable de type primitif, c'est la valeur de la variable qui est passée en paramètre
- Lorsque l'argument est un objet, il y a, théoriquement, deux éléments qui pourraient être passés en paramètre:
 - La référence vers l'objet
 - L'objet lui-même
- A la différence de C++, Java considère toujours que c'est la valeur de la référence (plus exactement une copie de cette valeur) et non l'objet qui est passée en argument

Exercices

- Comptes en banque
 - Créer une Classe banque
 - Transformer la classe banque en classe OO
 - Lui associer un tableau de comptes et un tableau de clients
 - Instancier quelques comptes et clients dans son constructeur et les associer les uns aux autres
 - **N'oubliez pas Faites d'abord un diagramme de classes!**
 - Toutes les opérations sur les comptes devront se faire obligatoirement au travers de la classe publique Banque (façade) – Les compteEnBanque et les Personne sont invisible en dehors du package!
 - Implémentez les méthodes : enregsitrerUnClient, ouvririComptePourClient, depotSurCompte, retraitreDeCompte, virement
 -

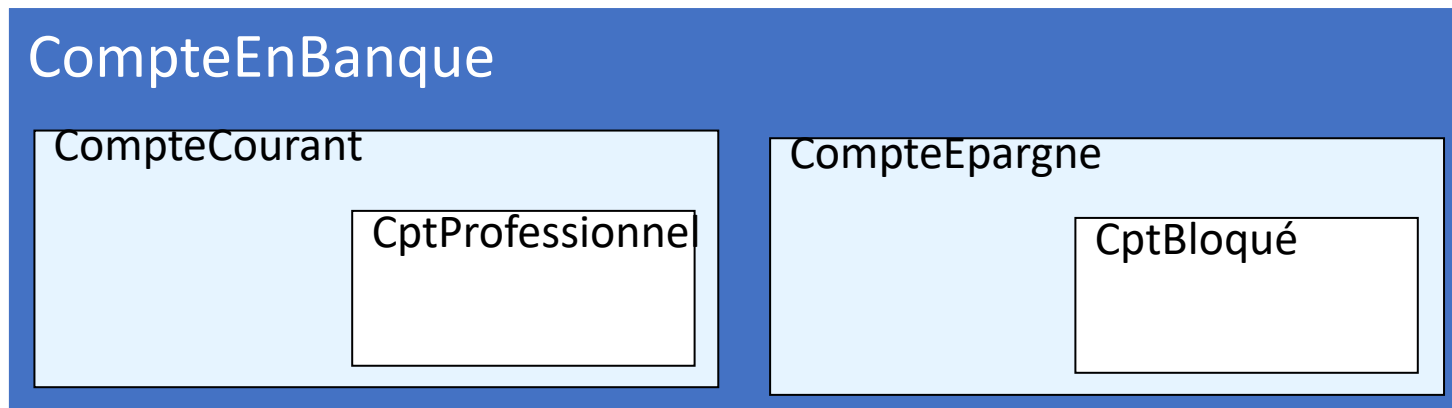
Exercices

- Bonus
 - Que se passe-t-il dans le cas de plusieurs banques?
 - Quelle conséquence cela-a-t-il sur les identifiants des comptes?
 - Comment assurer de manière transparente un virement interbancaire ?
 - Quelles solutions proposez-vous?

Héritage

En quoi consiste l'héritage?

- Supposons qu'il existe déjà une classe qui définit un certain nombre de messages et qu'on aie besoin d'une classe identique mais pourvue de quelques messages supplémentaires
 - ➔ Comment éviter de réécrire la classe de départ?
 - ➔ Regrouper les classes en super-classes en factorisant et spécialisant
 - ➔ La sous-classe hérite des attributs et méthodes et peut en rajouter de nouveaux

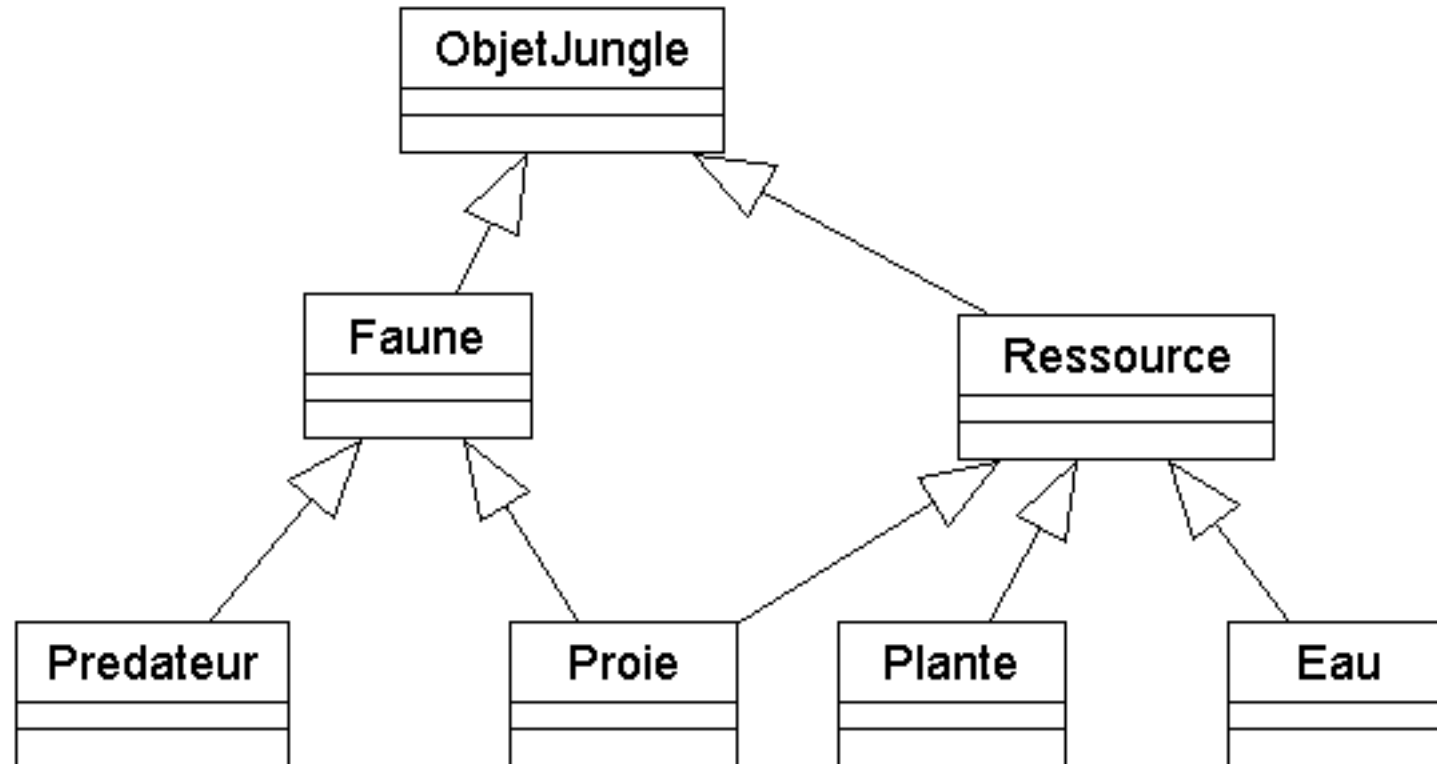


Héritage

En quoi consiste l'héritage?

• Ex: Dans l'écosystème

- La classe Faune regroupe les animaux
- La classe Ressource regroupe l'eau et la plante



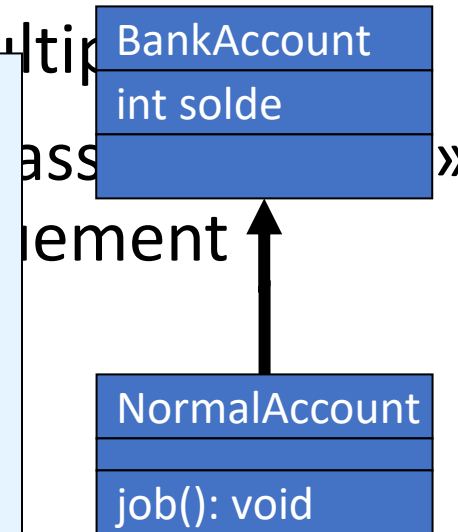
Héritage

Comment mettre en œuvre l'héritage?

- Pour réaliser un héritage en Java, il suffit de le déclarer dans la déclaration de la classe au moyen de la clause « *extends* »
- Une sous-classe hérite des variables et des méthodes de ses classes parentes

- Java n'offre pas la possibilité d'héritage multiple

```
class BankAccount {  
    protected int solde;  
    ...  
}  
  
class NormalAccount extends BankAccount {  
    public void job() {solde+=1000;}  
}
```



Héritage

Comment mettre en œuvre l'héritage?

- Comme une sous-classe hérite des variables de la classe parent, elle doit nécessairement les initialiser, donc appeler le constructeur de la classe parent
- Cela doit se faire avant la réalisation de son propre constructeur
- Concrètement, cela implique que la première instruction dans le constructeur d'une sous-classe doit être l'appel au constructeur parent
- Cet appel se fait au moyen du mot-clé « super »
- Si la classe n'y est pas, l'appel

```
class Child extends MyClass {  
    Child() {  
        super(6); // appel du constructeur  
        parent  
    }  
}
```

Héritage

Comment mettre en œuvre l'héritage?

- Le mot `super()`

```
class Employee {
    String name,firstname;
    Address a;
    int age;
    Employee(String name,String firstname,Address a,int
age){
        super(); // Comme le constructeur parent n'attend
pas
                // d'argument, cet appel n'est pas
obligatoire
        this.firstname=firstname;
        this.name=name;
        this.a=a;
        this.age=age;
    }
    Employee(String name,String firstname){
        this(name,firstname,null,-1);
    }
}
```

Héritage

Comment mettre en œuvre l'héritage?

- La variable `aNumber` du compte normal cache la variable `aNumber` de la classe générale compte en banque. Mais on peut accéder à la variable `aNumber` d'un compte en banque à partir d'un compte normal en utilisant le mot-clé `super` :

```
class BankAccount{  
    int aNumber;  
}  
  
class NormalAccount extends BankAccount{  
    float aNumber;  
}
```

Exercices

- Création d'une hiérarchie de compte
 - Distinguer les comptes courants et les livrets d'épargne
 - Quelle hiérarchie de classes pourrait-on proposer?
 - Quelles méthodes existeraient dans une sous-classe mais pas dans l'autre?
 - Implémenter cette hiérarchie dans le programme

Héritage

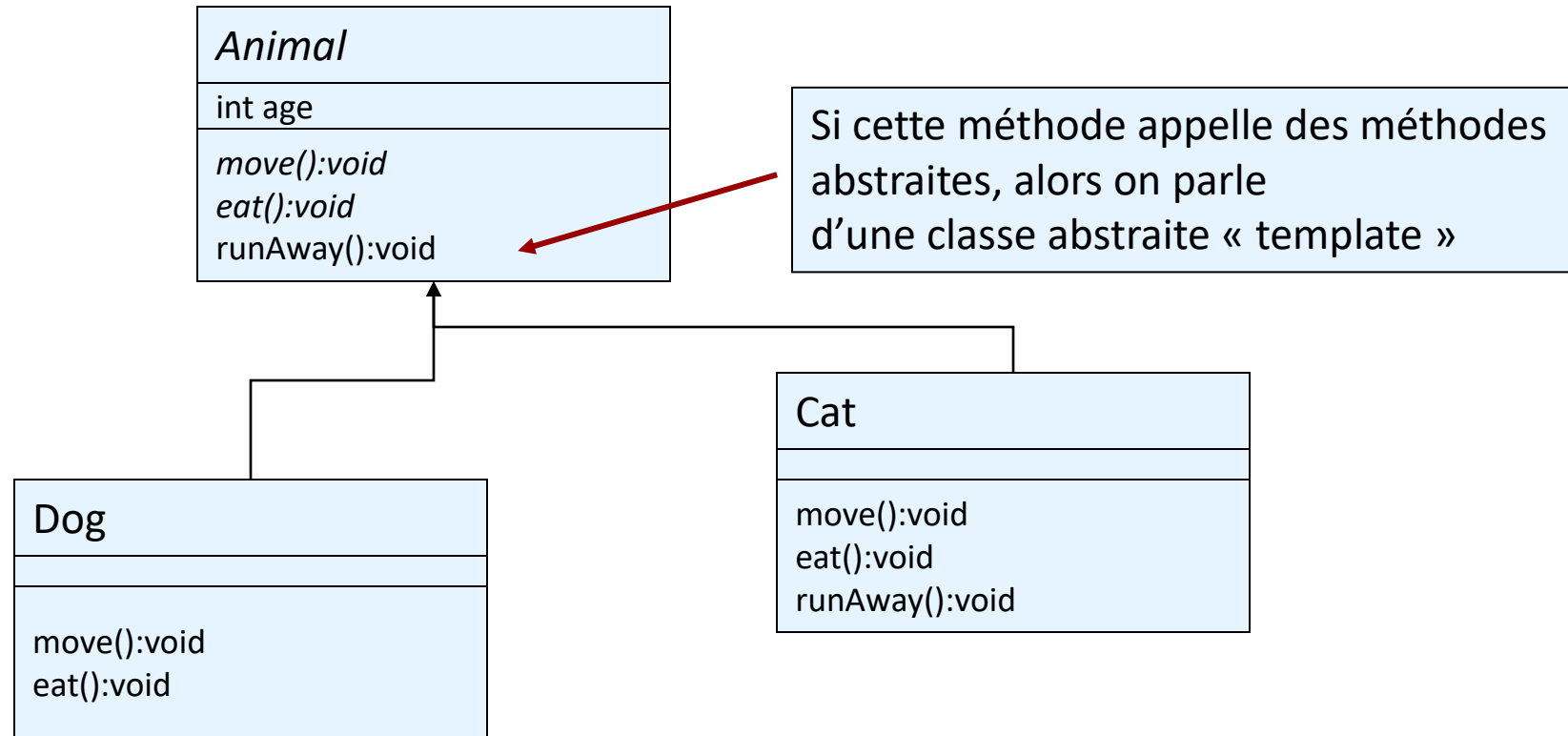
Comment mettre en œuvre l'héritage?

- Une classe abstraite
 - Peut contenir ou hériter de méthodes abstraites (des méthodes sans corps)
 - Peut contenir des attributs
 - Peut avoir des méthodes normales, avec corps
- Une classe abstraite ne peut être instanciée
 - On peut seulement instancier une sous-classe concrète
 - La sous-classe concrète doit donner un corps à toute méthode abstraite
 - En d'autres termes, on ne peut jamais faire un « new » sur une classe abstraite
 - En revanche on peut parfaitement créer des variables de types abstraits
- La déclaration d'une classe abstraite et d'une méthode abstraite ressemble à ceci:

```
abstract class Animal {  
    abstract void move();  
}
```


Héritage

Comment mettre en œuvre l'héritage?



Héritage

Comment mettre en œuvre l'héritage?

Mémoire des référents	Mémoire des valeurs	Mémoire des objets
Chien monChien	ab123d	
Animal IAnimalQueTuAsVu	ab123d	
Objet ceQuElleVientDeVoirPasser	ab123d	

Héritage

Comment mettre en œuvre l'héritage?

- Une méthode peut aussi être abstraite
 - Dans ce cas, elle ne peut pas contenir de corps (elle aura un “;” à la place des accolades)
- Si une classe comprend une méthode abstraite elle est obligatoirement abstraite elle-même
- Si une sous-classe hérite d'une classe abstraite, elle doit redéfinir toutes les méthodes abstraites de sa classe parent (leur donner un corps)
- Si elle ne le fait pas, elle sera condamnée à être abstraite elle-même

Exercices

- Création d'une hiérarchie de compte
 - Rendre la méthode afficheToi de la classe CompteEnBanque
 - Rendre la classe CompteEnBanque abstraite
 - Qu'est-ce que cela implique?

Exercices Bonus

- Ajouter un nouveau type de compte : un super compte
- Un super compte permet de gérer un ensemble de compte par lot. Chaque super compte est associés à un ensemble de compte en banque.
- Une opération sur un super compte est automatiquement redirigée vers les comptes qu'il contient.
- Exemple : Un dépôt de 1000 euros sur un Super compte qui contient 5 compte est réalisé par 5 dépôt de 250 euros sur les comptes contenus dans le super compte.
- Proposez un diagramme de classe

Conversion de types (1/2)

Définition • Java, langage fortement typé, impose le respect du type d'un objet

- Toutefois, il est possible de convertir le type d'un objet vers un type compatible
 - Un type A est compatible avec un type B si une valeur du type A peut être assignée à une variable du type B
 - Ex: Un entier et un double
- La conversion de type peut se produire
 - Implicitement (conversion automatique)
 - Explicitement (conversion forcée)
- La conversion explicite s'obtient en faisant précéder la variable du type vers lequel elle doit être convertie entre parenthèses (casting)

```
double d = 3.1416;  
int i = (int) d;
```

Conversion de types (2/2)

Application

- Appliquer un opérateur de « cast » au nom d'une variable
 - Ne modifie pas la valeur de la variable
 - Provoque le traitement du contenu de la variable en tant que variable du type indiqué, et seulement dans l'expression où l'opérateur de cast se trouve
- S'applique aux variables de types primitifs et aux variables de types de références
- Types primitifs:
 - Seulement vers un type plus large (ou risque de perte de données)
 - Interdit pour le type *boolean*
 - Ex: Short → Integer → Long
- Types de références:
 - Vers une classe parent ou une interface implémentée (ou risque d'erreur)
 - Dans ce cas, l'opérateur de cast n'est pas nécessaire (en fonction du contexte)
 - Peuvent toujours être castés vers OBJECT
 - Ex: Voiture → VéhiculesMotorisés → Véhicules → Object

Les interfaces (1/3)

Définition

- L'interface d'une classe = la liste des messages disponibles
= signature des méthodes de la classe
- Certaines classes sont conçues pour ne contenir précisément que la signature de leurs méthodes, sans corps. Ces classes ne contiennent donc que leur interface, c'est pourquoi on les appelle elles-mêmes *interface*
- Ne contient que la déclaration de méthodes, sans définition (corps)
- Permet des constantes globales

• Une classe peut implémenter une ou plusieurs interfaces

<pre>public interface Runnable { public abstract void run(); }</pre>	<pre>public interface GraphicalObject { public void draw(Graphics g); }</pre>
--	---

Les interfaces (2/3)

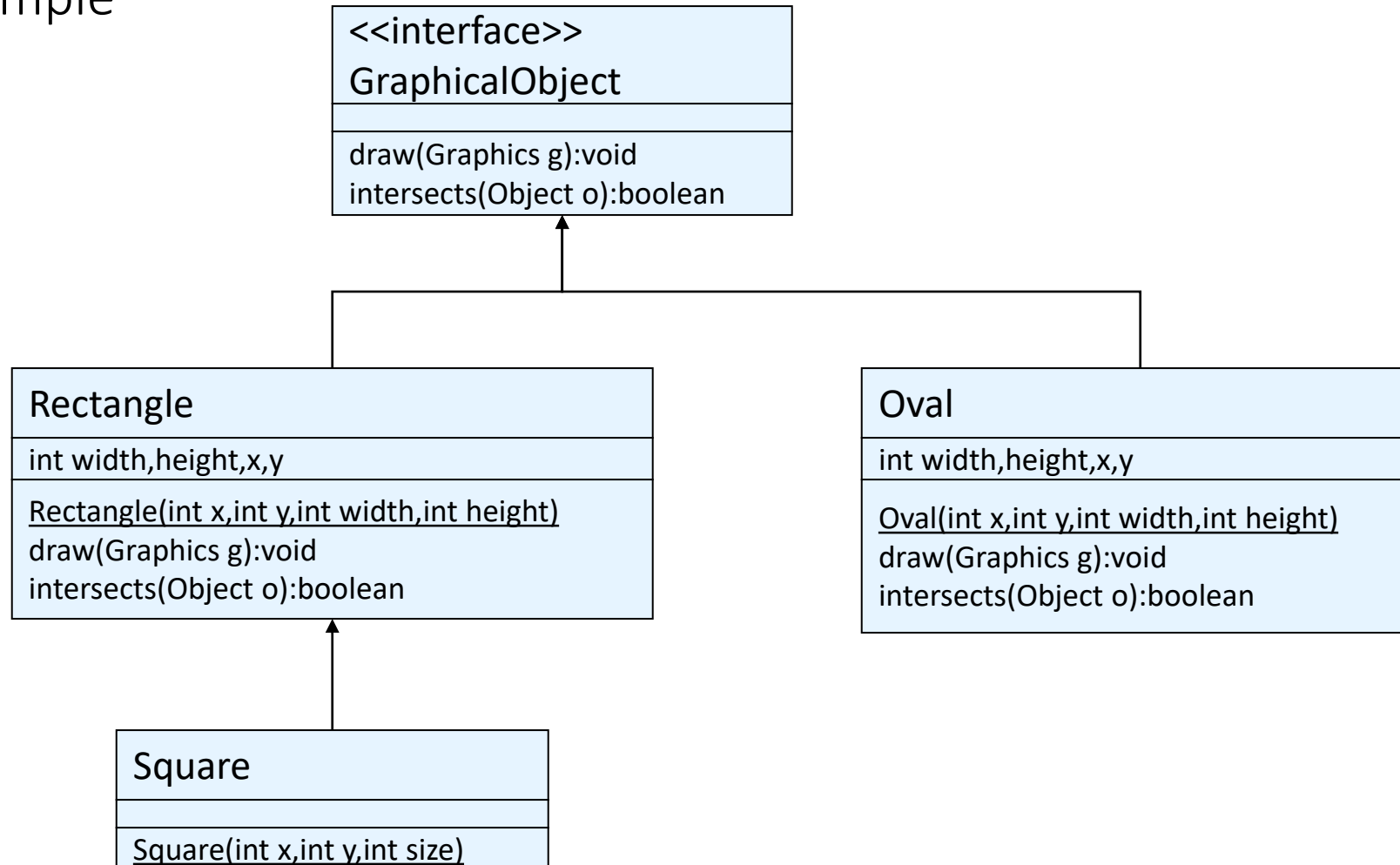
Raisons d'être

- Forcer la redéfinition / l'implémentation de ses méthodes
- Permettre une certaine forme de multi-héritage
- Faciliter et stabiliser la décomposition de l'application logicielle
- D'une classe qui dérive d'une interface, on dit qu'elle implémente cette interface
- Le mot clé associé est donc logiquement: `implements`
- Exemple:

```
public class monApplet extends Applet implements Runnable, KeyListener
```

Les interfaces (3/3)

Exemple



Introduction

Qu'est-ce qu'une collection?

- Collection
 - Un objet utilisé afin de représenter un groupe d'objets
- Utilisé pour:
 - Stocker, retrouver et manipuler des données
 - Transmettre des données d'une méthode à une autre
- Représente des unités de données formant un groupe logique ou naturel, par exemple:
 - Une main de poker (collection de cartes)
 - Un dossier d'emails (collection de messages)
 - Un répertoire téléphonique (collection de correspondances NOM – N°)

Introduction

Java Collections Framework

- Définit toute la structure des collections en Java
- Constitue une architecture unifiée pour
 - la représentation des collections
 - la manipulation des collections
- Contient des:
 - Interfaces
 - Types de données abstraits représentant des collections
 - Permettent de manipuler les collections indépendamment de leur représentation
 - Organisées en une hiérarchie unique
 - Implémentations
 - Implémentations concrètes des interfaces ➔ Structures de données réutilisables
 - Fournies pour accélérer le développement
 - Algorithmes
 - Méthodes standards fournissant des opérations comme le tri, la recherche, etc.

Interfaces

Structure

Il existe 4 groupes d'interfaces liées aux collections

- Collection
 - Racine de la structure des collections
 - Représente un groupe d'objets (dits « éléments »)
 - Peut autoriser ou non les duplicats
 - Peut contenir un ordre intrinsèque ou pas
- Map
 - Un objet qui établit des correspondances entre des clés et des valeurs
 - Ne peut en aucun cas contenir des duplicats
 - ➔ Chaque clé ne peut correspondre qu'à une valeur au plus
- Interfaces de comparaison
 - Permettent le tri des objets du type implémentant l'interface
 - Deux versions: « Comparator » et « Comparable »
- Iterator
 - Permet de gérer les éléments d'une collection

Interfaces

Collection (1/2)

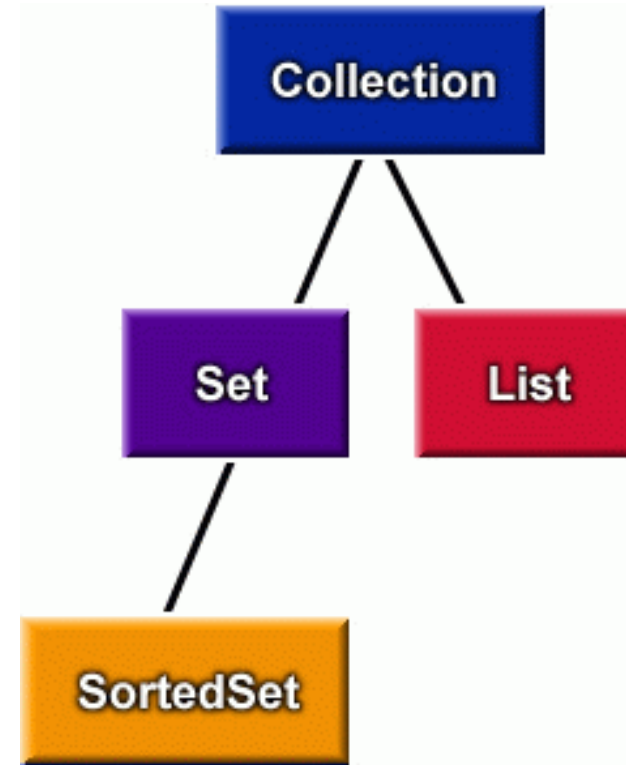
- La racine de la structure des collections
 - Sans ordre spécifique
 - Duplicats permis
- Définit les comportements standards des collections
 - Vérification du nombre d'éléments
 - ➔ *size()*, *isEmpty()*
 - Test d'appartenance d'un objet à la collection
 - ➔ *contains(Object)*
 - Ajout et suppression d'éléments
 - ➔ *add(Object)*, *remove(Object)*
 - Fournir un itérateur pour la collection
 - ➔ *iterator()*
 - Bulk Operations
 - ➔ *addAll(Collections)*, *clear()*, *containsAll(Collections)*

Interfaces

Collection (2/2)

Trois variantes principales:

- Set
 - Duplicats interdits
 - Sans ordre spécifique
- List
 - Duplicats permis
 - Contient un ordre spécifique intrinsèque
 - Parfois appelé « Séquences »
 - Permet 2 méthodes d'accès particulières:
 - Positional Access: manipulation basée sur l'index numérique des éléments
 - Search: recherche d'un objet en particulier dans la liste et renvoi de son numéro d'index (sa position dans la liste)
- SortedSet
 - Duplicats interdits
 - Contient un ordre spécifique intrinsèque



Interfaces

- Map
- Map = Objet qui contient des correspondances Clé – Valeur
 - Ne peut contenir de doublons (clés sont primaires)
 - Fournit des opérations de base standards:
 - `put(key,value)`
 - `get(key)`
 - `remove(key)`
 - `containsKey(key)`
 - `containsValue(value)`
 - `size()`
 - `isEmpty()`
 - Peut générer une collection qui représente les couples Clé – Valeur
 - Il existe aussi
 - des « Map » permettant de faire correspondre plusieurs valeurs à chaque clé
 - des « SortedMap » fournissant des « Map » naturellement triés

Interfaces

Iterator Permet de gérer les éléments d'une collection

- Toute collection peut fournir son « Iterator »
- Permet également de supprimer des éléments d'une collection au cours d'une phase d'itération sur la collection
- Contient 3 méthodes essentielles:
 - `hasNext(): boolean` → Indique s'il reste des éléments après l'élément en cours
 - `next(): Object` → Fournit l'élément suivant de la collection
 - `Remove(): void` → Supprime l'élément en cours
- Exemple:

```
static void filter(Collection c) {  
    for (Iterator i = c.iterator(); i.hasNext(); )  
        if (! cond(i.next()))  
            i.remove();  
}
```

Interfaces

Comparaison (1/4) Deux interfaces

- Comparable
 - Fournit une méthode de comparaison au sein d'une classe
 - Impose de redéfinir une seule méthode `public int compareTo(Object o)` qui renvoie un entier:
 - 1 si l'objet courant > l'objet « o » fourni dans la méthode
 - 0 si l'objet courant = l'objet « o » fourni dans la méthode
 - 1 si l'objet courant < l'objet « o » fourni dans la méthode
- Comparator
 - Permet de définir une classe servant de comparateur à une autre
 - Impose de redéfinir une seule méthode `public int compare(Object o1, Object o2)` qui renvoie un entier:
 - 1 si `o1 > o2`
 - 0 si `o1 = o2`
 - 1 si `o1 < o2`

Interfaces

Comparaison (2/4)

Exemple d'implémentation de *Comparable*

```
import java.util.*;
public class Name implements Comparable {
    private String  firstName, lastName;

    public Name(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public int compareTo(Object o) {
        Name n = (Name) o;
        int lastCmp = lastName.compareTo(n.lastName);
        if(lastCmp==0)
            lastCmp = firstName.compareTo(n.firstName);
        return lastCmp;
    }
}
```


Interfaces

Comparaison (3/4)

Depuis Java 1.5, l'interface peut être générique:

```
import java.util.*;

public class Personne implements Comparable<Personne> {
    private String  firstName, lastName;

    public Personne(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public int compareTo(Personne n) {
        int lastCmp = lastName.compareTo(n.lastName);
        if(lastCmp==0)
            lastCmp = firstName.compareTo(n.firstName);
        return lastCmp;
    }
}
```

Interfaces

Comparaison (4/4)

- Les types primitifs

Class	Natural Ordering
Byte	signed numerical
Character	unsigned numerical
Long	signed numerical
Integer	signed numerical
Short	signed numerical
Double	signed numerical
Float	signed numerical
BigInteger	signed numerical
BigDecimal	signed numerical
File	system-dependent lexicographic on pathname.
String	lexicographic
Date	chronological

de naturel

Exception

- Quand un cas non prévu survient, il est possible de le capter et le traiter par le mécanisme d'*Exception*
 - Si on capte et traite une exception, le programme peut continuer à se dérouler
 - Sinon, le programme sort de l'exécution avec un message d'erreur
 - Exemple d'exception: division par 0, ouvrir un fichier qui n'existe pas, ...
- Mécanisme de traitement d'exception
 - Définir des classes d'exception
 - *Exception*
 - *IOException* input output exeption
 - *EOFException*, ... End of file
 - Utiliser *try-catch* pour capter et traiter des exceptions

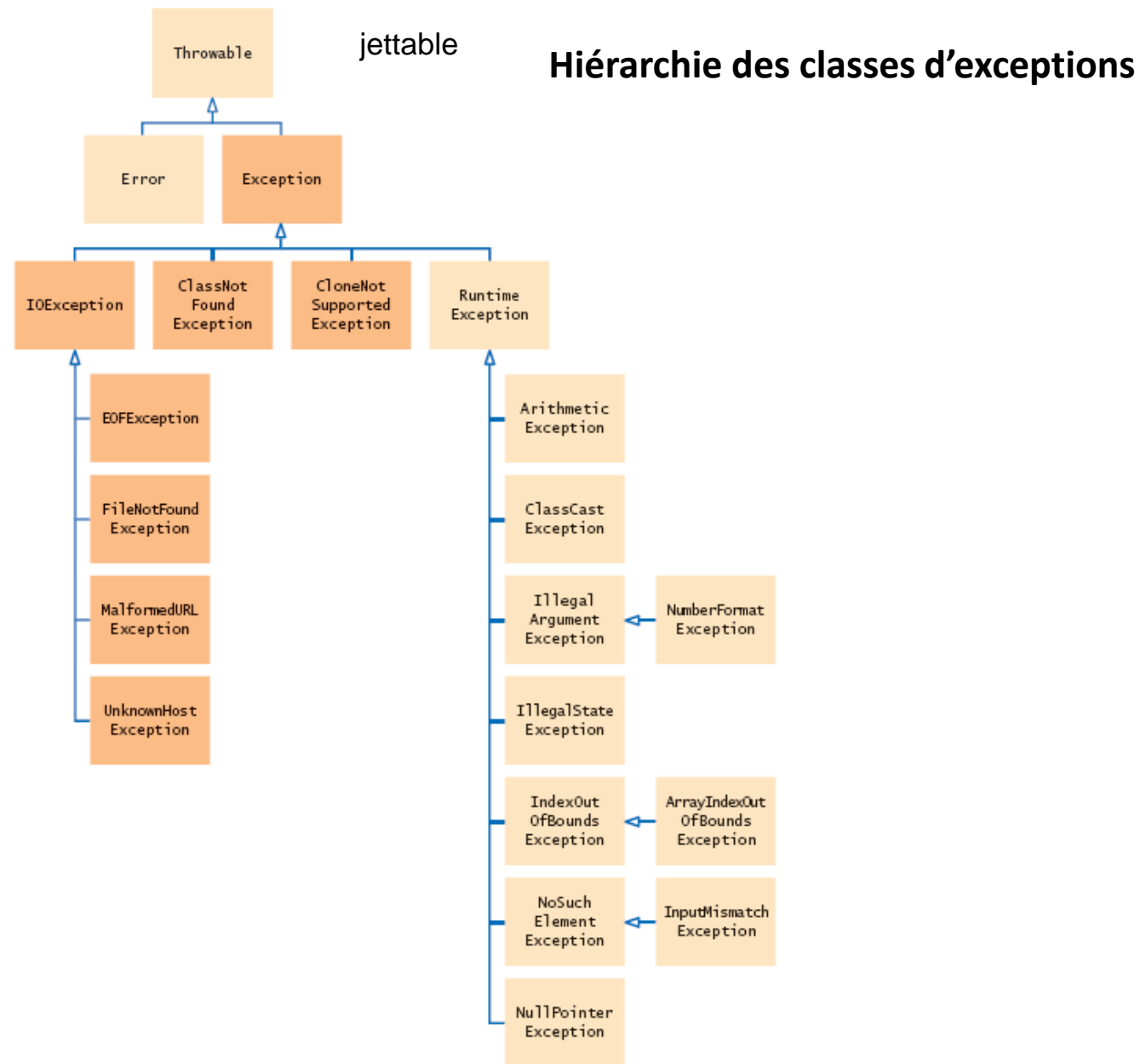


Figure 1 The Hierarchy of Exception Classes

Attraper (*catch*) une exception

- Attraper une exception pour la traiter

```
try {  
    statements  
    :  
    :  
} catch (ExceptionClass1 object) {  
    statements  
    :  
    :  
} catch (ExceptionClass2 object) {  
    statements  
    :  
    :  
} ...
```

Bloc où une exception
peut se générer

Blocs pour attraper
les exceptions

Exemple

```
public static void ouvrir_fichier(String nom) {  
    try {  
        input = new BufferedReader(new FileReader(nom));  
    }  
    catch (IOException e) {  
        System.err.println("Impossible d'ouvrir le fichier d'entree.\n" +  
                             e.toString());  
        System.exit(1);  
    }  
}
```

ouverture d'un fichier

try: on tente d'effectuer des opérations

catch: si une exception de tel type survient au cours, on la traite de cette façon

finally

C'est un bloc d'instruction qui sera toujours exécuté...toujours!!!

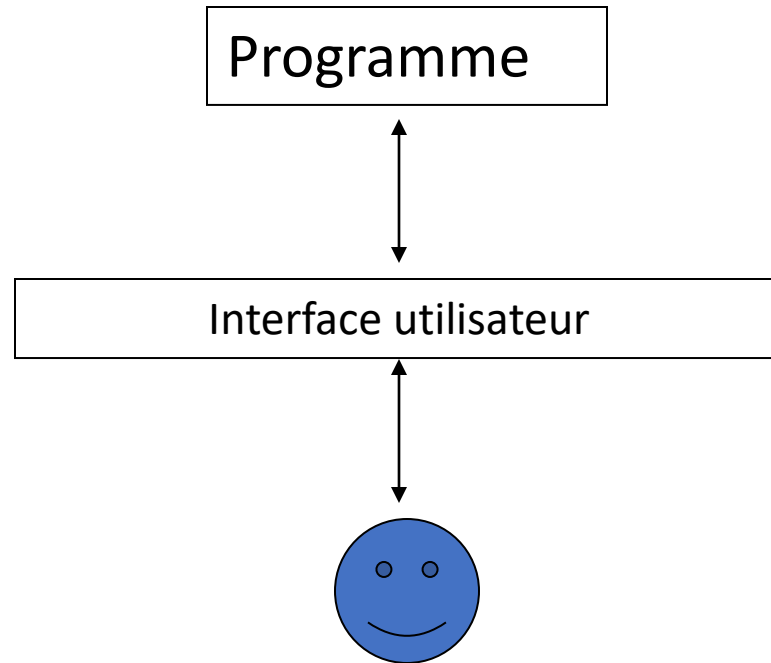
- Souvent combiné avec catch

```
try
{
    statements
    ...
}
catch (ExceptionClass exceptionObject)
{
    statements
    ...
}
finally
{
    statements
    ...
}
```

Même si une exception est
attrapée, *finally* sera toujours
exécuté

Utile pour s'assurer de certaine
sécurité (*cleanup*)

Généralité



Rôles d'une interface utilisateur:

- montrer le résultat de l'exécution
- permettre à l'utilisateur d'interagir

Exemple simple

```
import javax.swing.*;
```

Importer le package

```
public class DisplayFrame {
```

```
    public static void main (String[]
```

```
        JFrame f = new JFrame("FrameDe
```

```
        //... components are added to its content frame.
```

```
        f.setSize(300,200);
```

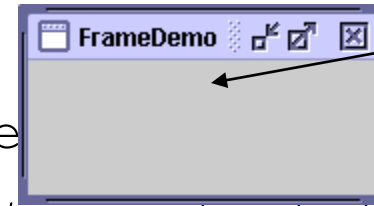
```
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        f.setVisible(true);
```

```
    }
```

```
}
```

afficher



Créer un
objet

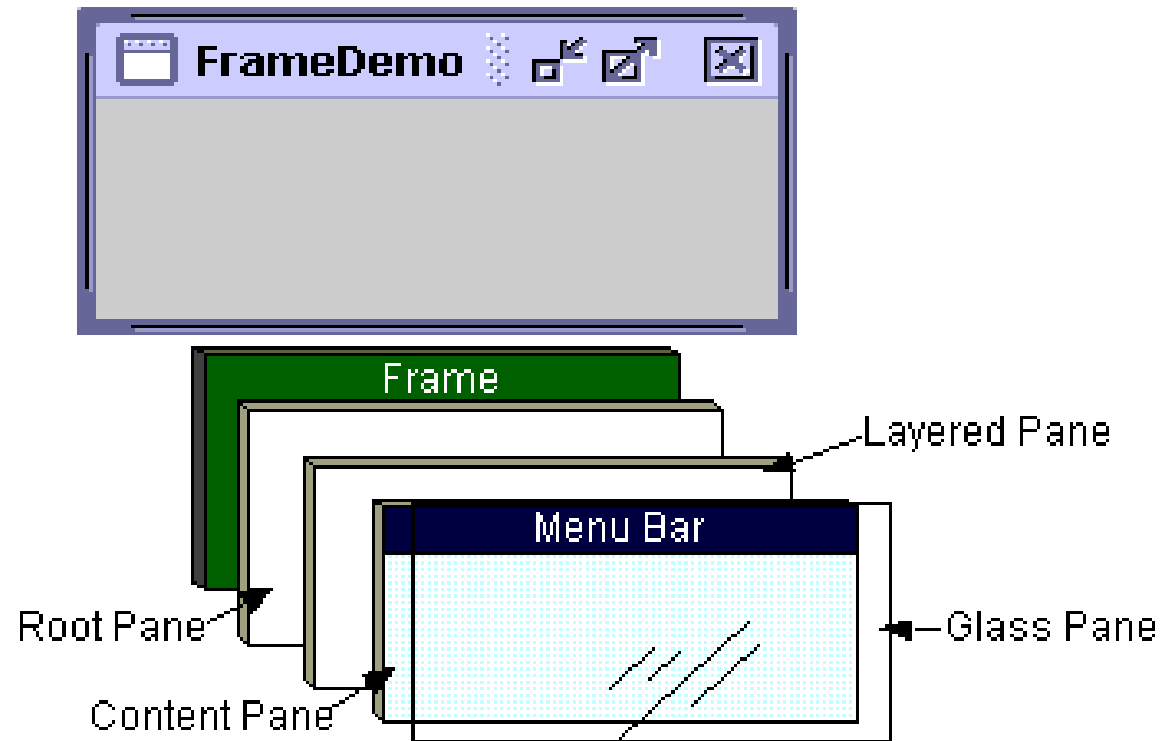
Définir la taille

Afficher une interface

- Importer le package (les classes)
 - Les classes sont regroupées en package
 - Importer un package = importer toutes les classes du package
 - `import javax.swing.*;`
- Créer une fenêtre graphique (JFrame, ...)
- Mettre les paramètres (taille, ...)
- Afficher
- Différence:
 - `import java.awt.*;` les classes dans *awt*
 - `import java.awt.event.*;` les classes dans *event*

Insérer des éléments dans la fenêtre

- Composition d'une fenêtre JFrame



Structure interne de JFrame

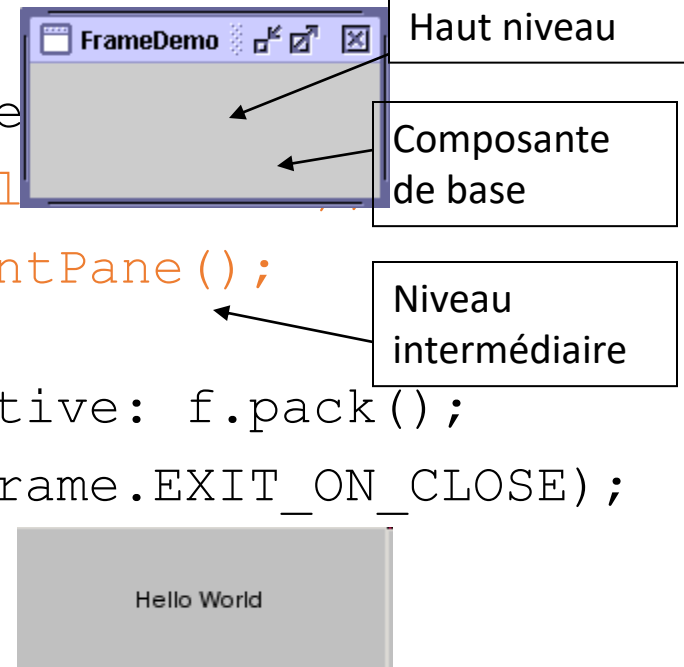
Typiquement, on insère des éléments graphiques dans ContentPane

Ajouter des composants dans une fenêtre

```
import javax.swing.*;

public class DisplayFrame {

    public static void main (String[] args) {
        JFrame f = new JFrame("FrameDemo");
        JLabel label = new JLabel("Hello World");
        JPanel p = (JPanel)f.getContentPane();
        p.add(label);
        f.setSize(300,200); //alternative: f.pack();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
}
```



Composer une fenêtre

- Créer une fenêtre (1)
- Créer un ou des composants intermédiaires (2)
 - Pour *JFrame*, un *JPanel* est associé implicitement (ContentPane)
- Créer des composants de base (3)
- Insérer (3) dans (2)
- Insérer (2) dans (1)
- Afficher

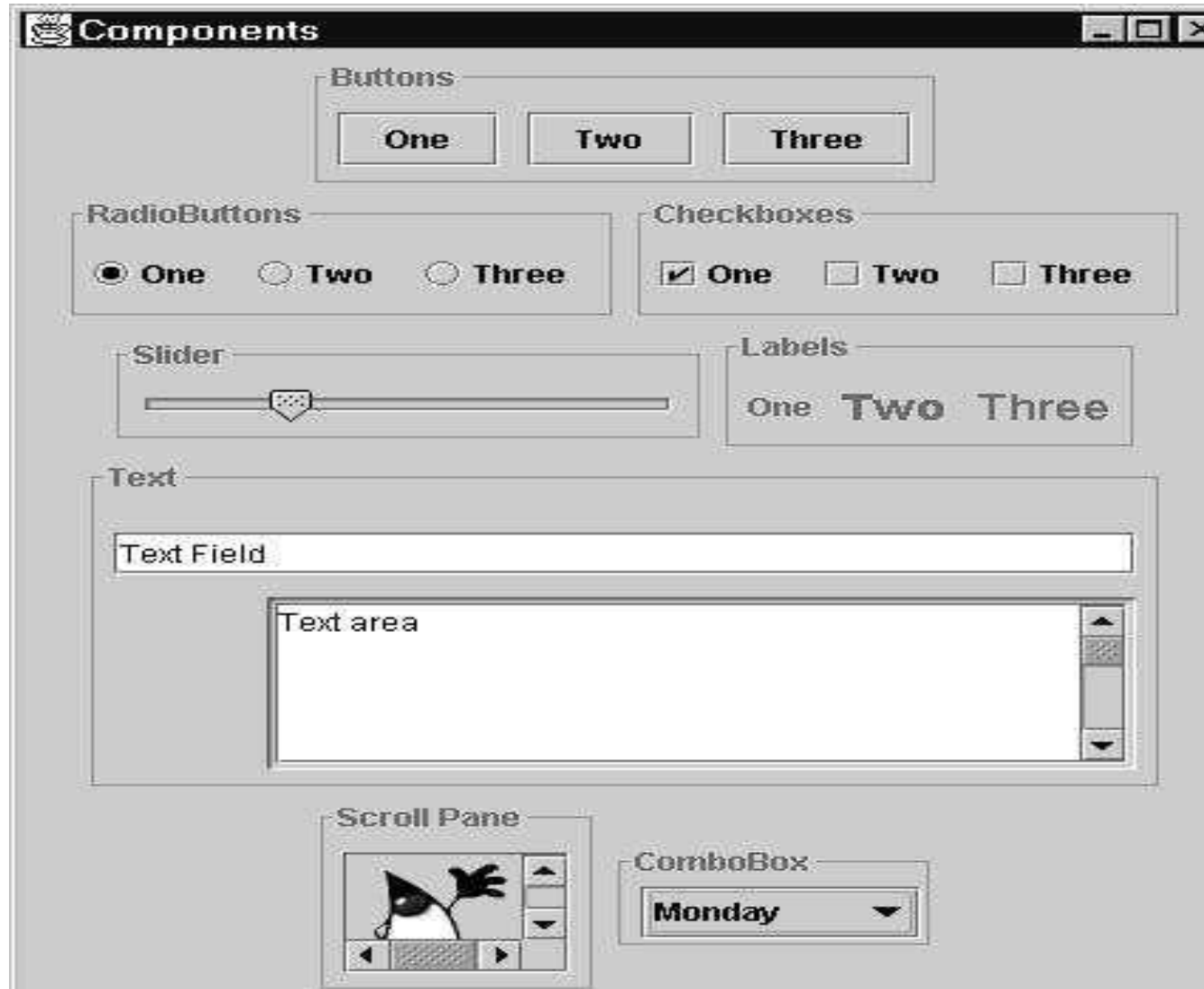
Composants de base pour obtenir des données

- *JButton*
- *JCheckBox* a toggled on/off button displaying state to user.
- *JRadioButton* a toggled on/off button displaying its state to user.
- *JComboBox* a drop-down list with optional editable text field. The user can key in a value or select a value from drop-down list.
- *Jlist* allows a user to select one or more items from a list.
- *Jmenu* popup list of items from which the user can select.
- *Jslider* lets user select a value by sliding a knob.
- *JTextField* area for entering a single line of input.

Composants de base pour afficher l'information

- *JLabel* contains text string, an image, or both.
- *JProgressBar* communicates progress of some work.
- *JToolTip* describes purpose of another component.
- *Jtree* a component that displays hierarchical data in outline form.
- *Jtable* a component user to edit and display data in a two-dimensional grid.
- *JTextArea*, *JTextPane*, *JEditorPane*
 - define multi-line areas for displaying, entering, and editing text.

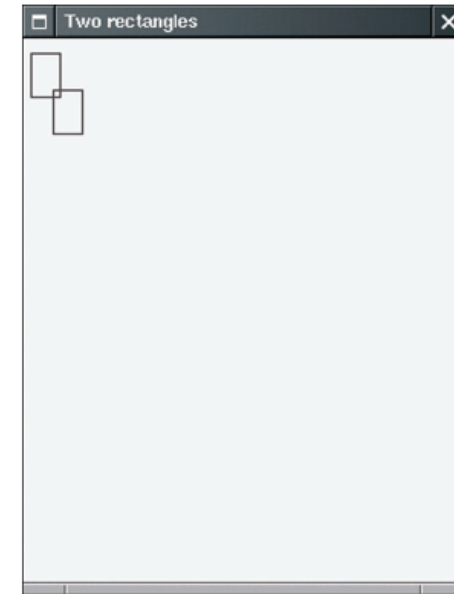
Swing components: Illustration



Définir ses propres classes

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import java.awt.Rectangle;
04: import javax.swing.JPanel;
05: import javax.swing.JComponent;
06:
07: /**
08:  A component that draws two rectangles.
09:  */
10: public class RectangleComponent extends JComponent
11: {
12:     public void paintComponent (Graphics g)
13:     {
14:         // Recover Graphics2D
15:         Graphics2D g2 = (Graphics2D) g;
16:
17:         // Construct a rectangle and draw it
18:         Rectangle box = new Rectangle(5, 10, 20, 30);
19:         g2.draw(box);
20:
21:         // Move rectangle 15 units to the right and 25 units down
22:         box.translate(15, 25);
23:
24:         // Draw moved rectangle
25:         g2.draw(box);
26:     }
27: }
```

Figure 2
Drawing Rectangles



Créer et voir l'objet

```
01: import javax.swing.JFrame;
02:
03: public class RectangleViewer
04: {
05:     public static void main(String[] args)
06:     {
07:         JFrame frame = new JFrame();
08:
09:         final int FRAME_WIDTH = 300;
10:         final int FRAME_HEIGHT = 400;
11:
12:         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
13:         frame.setTitle("Two rectangles");
14:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15:
16:         RectangleComponent component = new RectangleComponent();
17:         frame.add(component);
18:
19:         frame.setVisible(true);
20:     }
21: }
```

Alternative de JFrame: JApplet

- Applet = une fenêtre dans un navigateur
- Permet à n'importe quel utilisateur de lancer une application
- Plus de contrainte de sécurité (pas d'écriture)
- Programme englobé dans une page Web

Afficher deux Rectangles

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import java.awt.Rectangle;
04: import javax.swing.JApplet;
05:
06: /**
07:  * An applet that draws two rectangles.
08:  */
09: public class RectangleApplet extends JApplet
10: {
11:     public void paint (Graphics g)
12:     {
13:         // Prepare for extended graphics
14:         Graphics2D g2 = (Graphics2D) g;
15:
16:         // Construct a rectangle and draw it
17:         Rectangle box = new Rectangle(5, 10, 20, 30);
18:         g2.draw(box);
19:
20:         // Move rectangle 15 units to the right and 25 units down
21:         box.translate(15, 25);
22:
23:         // Draw moved rectangle
24:         g2.draw(box);
25:     }
26: }
27:
```

Au lancement, paint(...) est automatiquement exécutée
Pour ré-exécuter: repaint()

Lancer un Applet

À partir d'une page Web:

```
<html>
  <head>
    <title>Two rectangles</title>
  </head>
  <body>
    <p>Here is my <i>first applet</i>:</p>
    <applet code="RectangleApplet.class" width="300" height="400">
    </applet>
  </body>
</html>
```

Différence

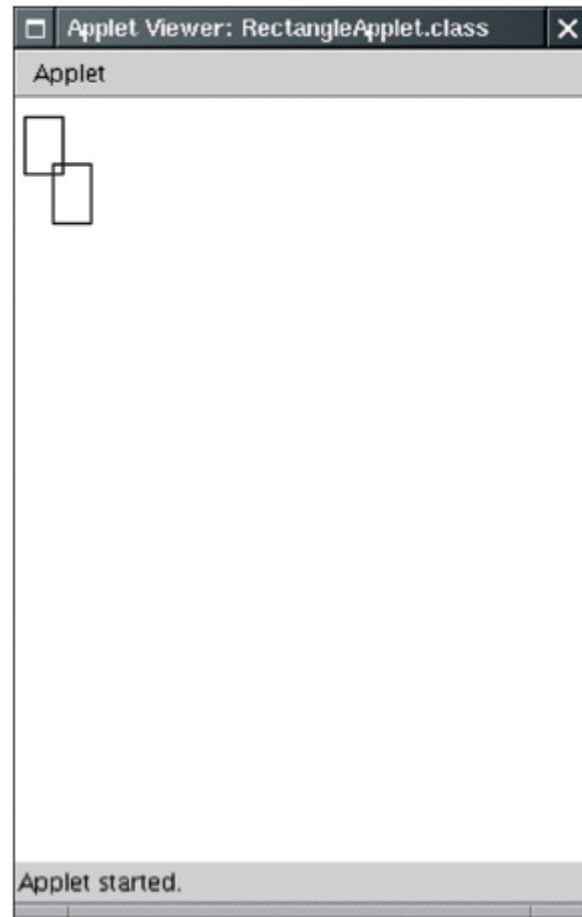


Figure 3
An Applet in the Applet Viewer

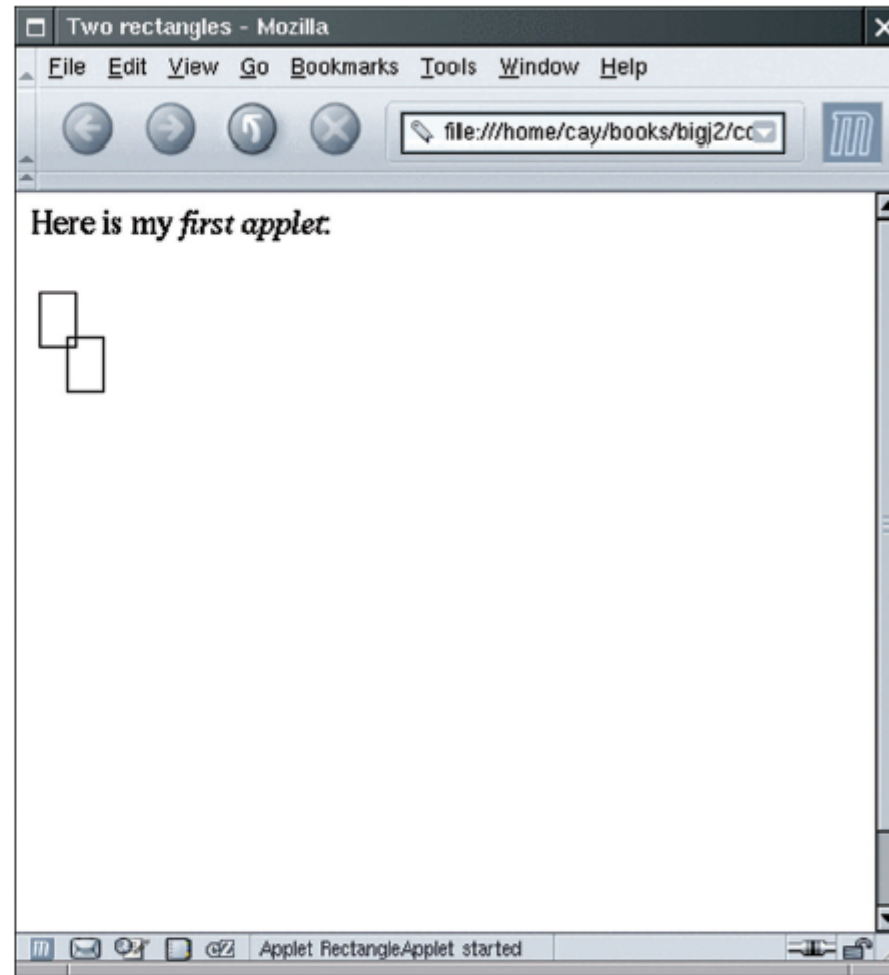


Figure 4
An Applet in a Web Browser

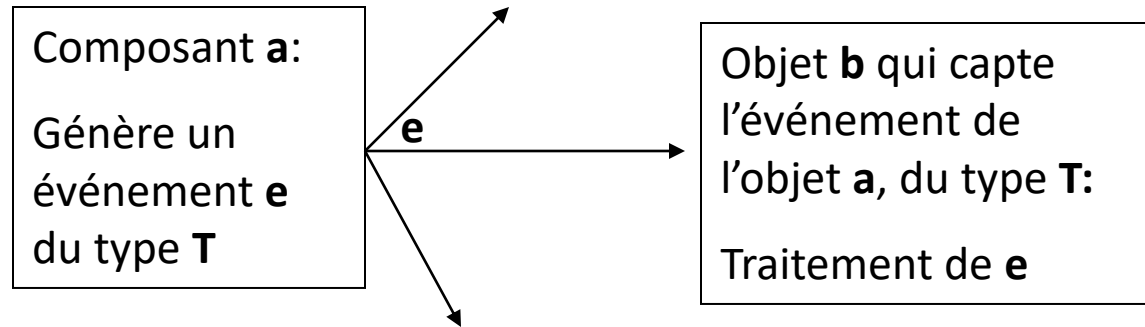
Événement

- Les événements sont des objets
- Sous-classes de la class abstraite `java.awt.AWTEvent`
- Les composants génèrent des événements
- Événements: chaque interaction de l'utilisateur sur un composant génère un événement
 - bouger la souris
 - cliquer sur un bouton
 - fermer une fenêtre
 - ...
- Un événement contient des informations: source, type d'événement, ...

```
public Object getSource();
```

- Utile pour détecter d'où provient l'événement

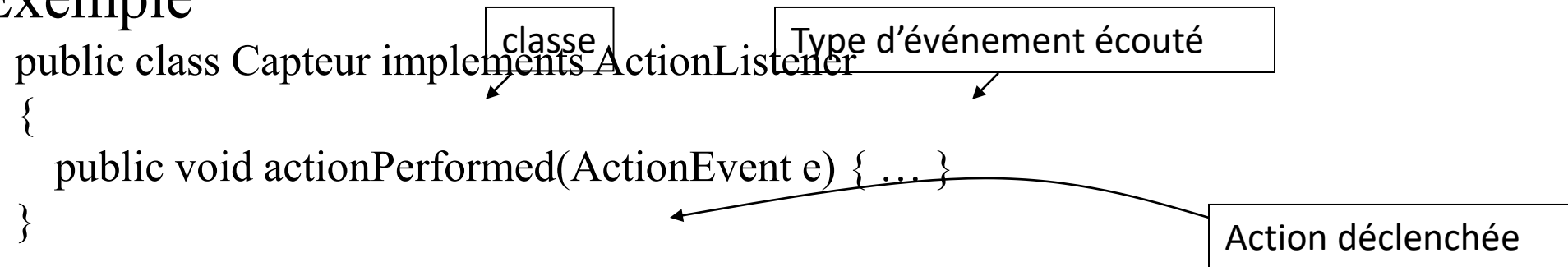
Propagation et traitement des événements



- Les événements sont générés et propagés
- Certains autres objets sont capables de capter des événements des types spécifiés, provenant de ces composants
 - **b** écoute les événements du type **T** venant de **a**
 - **b** est un **listener** de **a**
- On peut activer le traitement suite à la capture d'un événement
 - Le traitement lancé par l'objet **b**
- Programmation par événement
 - Le programme réagit aux événements

Listener et Event handler: donner la capacité d'entendre un événement

- *Listener*: Un objet est intéressé à *écouter* l'événement produit (être signalé quand il y a un événement)
- *Listener* doit implanter l'interface *event listener interface* associée à chaque type d'événement
- *Event Handler*: le programme qui lance un traitement suite à un événement
- Exemple



Exemple: changer la couleur

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class JAppletExample
    extends JApplet
    implements ActionListener
{
    private JButton rouge, bleu;
    private Color couleur = Color.BLACK;

    public void init()
    {
        rouge = new JButton("Rouge");
        bleu = new JButton("Bleu");

        Container content = getContentPane();
        content.setLayout(new FlowLayout());
        content.add(rouge);
        content.add(bleu);
    }
}
```

```
// Liens d'ecoute
    rouge.addActionListener(this);
    bleu.addActionListener(this);
}

// affichage
    public void paint(Graphics g)
    {
        super.paint(g);
        g.setColor(couleur);
        g.drawString("Choisir une couleur.", 100, 100);
    }

// methode qui reagit aux evenements
    public void actionPerformed (ActionEvent e)
    {
        if (e.getSource() == rouge) couleur=Color.RED;
        else if (e.getSource() == bleu) couleur = Color.BLUE;
        repaint(); //appeler paint(...) pour repaindre
    }
}
```

Types d'événements et écouteur

- **ActionEvent, ActionListener:**
 - Button, List, TextField, MenuItem, JButton, ...
 - `public void actionPerformed(ActionEvent)`
- **AdjustmentEvent, AdjustmentListener**
 - Scrollbar, ScrollPane, ...
 - `public void adjustmentValueChanged(AdjustmentEvent)`
- **ItemEvent, ItemListener**
 - Checkbox, CheckboxMenuItem, Choice, List
 - `public void itemStateChanged(ItemEvent)`

Types d'événements et écouteur

- MouseEvent
 - Souris
 - MouseListener
 - `public void mouseDragged(MouseEvent)`
 - `public void mouseMoved(MouseEvent)`
 - MouseMotionListener
 - `public void mousePressed(MouseEvent)`
 - `public void mouseReleased(MouseEvent)`
 - `public void mouseEntered(MouseEvent)`
 - `public void mouseExited(MouseEvent)`
 - `public void mouseClicked(MouseEvent)`
- TextEvent, TextListener
 - TextComponent et ses sous-classes
 - `public void textValueChanged(TextEvent)`

Télécommunication en Java

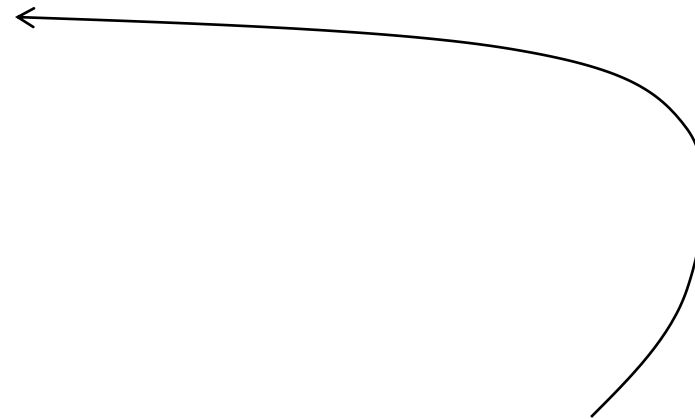
- Communication sur l'Internet
- Connexion dans Java

Internet

- Stockage de données (informations)
 - Serveur
 - Client
- Connexion
 - Connexion entre un client et un serveur
 - Un canal de communication
- Transmission
 - Protocole:
 - définit les commandes
 - le format de données transmises

Schéma de communication typique

- Serveur:
 - Il est lancé, en attente de recevoir un message (commande)
- Client
 - Demande à établir une connexion avec le serveur
 - Transmet une commande au serveur
- Serveur
 - Reçoit la commande
 - Traite la commande
 - Renvoie la réponse
- Client
 - Reçoit la réponse
 - Continue à traiter, transmet une autre commande, ...



Établir une connexion

- Identifier l'adresse du serveur à laquelle envoyer une requête de connexion
- Adresse:
 - Adresse IP (Internet Protocol): 4 octets (4 entiers 0-255)
 - 130.65.86.66
- *Domain Naming Service* (DNS): le nom correspondant à une adresse IP
 - Ss_domaine. sous_domaine . domaine
 - java.sun.com, www.iro.umontreal.ca
 - Traduction de DNS en adresse IP: par un serveur DNS
- Serveur
 - Prêt à recevoir des requêtes des types préétablis
 - E.g. GET

Protocole

- Un serveur est établi pour communiquer selon un protocole
- Canal de communication (numéro de port)
 - 0 and 65,535
 - HTTP: par défaut: 80
- Serveur Web: prêt à recevoir les requêtes HTTP:
 - Adresse d'un document:
 - Uniform Resource Locator (URL)
 - java.sun.com/index.html
 - Commande
 - GET /index.html HTTP/1.0 (suivie d'une ligne blanche)
 - http://java.sun.com/index.html

Protocole HTTP

Commande	Signification
• GET	Return the requested item
• HEAD	Request only the header information of an item
• OPTIONS	Request communications options of an item
• POST	Supply input to a server-side command and return the result
• PUT	Store an item on the server
• DELETE	Delete an item on the server
• TRACE	Trace server communication

En Java

- Établir une connexion avec un serveur Web
 - Créer un socket entre Client et Serveur
 - `Socket s = new Socket(hostname, portnumber);`
 - `Socket s = new Socket("java.sun.com", 80);`

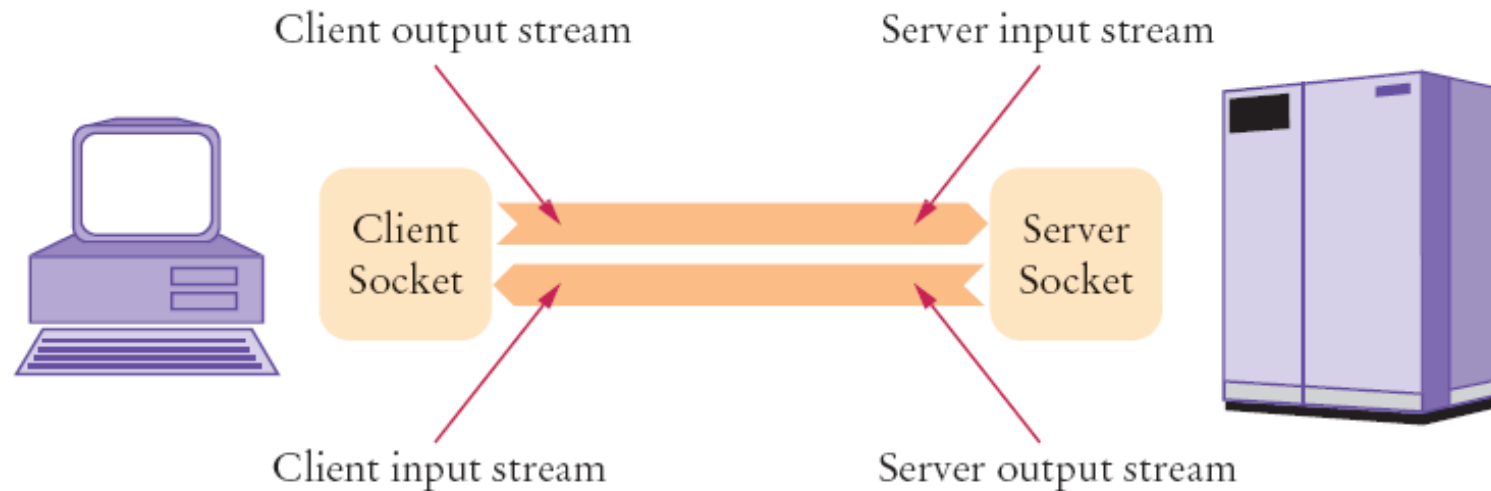


Figure 5 Client and Server Sockets

En Java

- Obtenir les *streams* du *socket*

```
InputStream instream = s.getInputStream();
```

```
OutputStream outstream = s.getOutputStream();
```

- *Cast* les *streams*

```
Scanner in = new Scanner(instream);
```

```
PrintWriter out = new PrintWriter(outstream);
```

- Fermer le *socket*

```
s.close();
```

Exemple

- Un programme pour obtenir une page web d'un site
 - établir une connexion avec un serveur
 - envoyer une requête
 - recevoir la réponse
 - fermer

```
java WebGet java.sun.com /
```

- Lancer WebGet avec 2 paramètres:
 - java.sun.com: DNS
 - /: page racine
 - Port par défaut: 80

```

01: import java.io.InputStream;
02: import java.io.IOException;
03: import java.io.OutputStream;
04: import java.io.PrintWriter;
05: import java.net.Socket;
06: import java.util.Scanner;
07:
14: public class WebGet
15: {
16:     public static void main(String[] args) throws IOException
17:     {
18:
19:
20:         String host;
21:         String resource;
22:
23:         if (args.length == 2)
24:         {
25:             host = args[0];
26:             resource = args[1];
27:         }
28:         else
29:         {
30:             System.out.println("Getting / from java.sun.com");
31:             host = "java.sun.com";
32:             resource = "/";
33:         }
34:

```

```

37:         final int HTTP_PORT = 80;
38:         Socket s = new Socket(host, HTTP_PORT);
39:
42:         InputStream instream = s.getInputStream();
43:         OutputStream outstream = s.getOutputStream();
44:
47:         Scanner in = new Scanner(instream);
48:         PrintWriter out = new PrintWriter(outstream);
49:
52:         String command = "GET " + resource + " HTTP/1.0\n\n";
53:         out.print(command);
54:         out.flush();
55:
58:         while (in.hasNextLine())
59:         {
60:             String input = in.nextLine();
61:             System.out.println(input);
62:         }
63:
66:         s.close();
67:     }
68: }

```


Résultat: java WebGet

```
HTTP/1.1 200 OK
Server: Sun-Java-System-Web-Server/6.1
Date: Tue, 28 Mar 2006 20:07:26 GMT
Content-type: text/html; charset=ISO-8859-1
Set-Cookie: SUN_ID=132.204.24.63:218361143576446; EXPIRES=Wednesday, 31-Dec-2025
23:59:59 GMT; DOMAIN=.sun.com; PATH=/
Set-cookie: JSESSIONID=519A024C45B4C300DA868D076CA33448; Path=/
Connection: close
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.or
g/TR/html4/loose.dtd">
<html>
<head>
<title>Java Technology</title>
<meta name="keywords" content="Java, platform" />
<meta name="collection" content="reference">
<meta name="description" content="Java technology is a portfolio of products tha
t are based on the power of networks and the idea that the same software should
run on many different kinds of systems and devices." />
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
<meta name="date" content="2006-03-23" />
...
```

Résumé sur Java

- Programmation classique: traiter des données des types primitifs
- Programmation OO: regrouper les données (attributs) et leurs traitements (méthodes)
- Outils disponibles
 - Classes regroupées dans des packages
 - Interface graphique
 - Communication à travers l'Internet
 - Package pour interagir avec un SGBD (Système de gestion de base de données)
 - ...

À retenir

- Programme = ?
- Comment un programme est traduit en code exécutable? (compilation ou interprétation)
- Environnement de programmation
- Quels sont les opérations qu'on peut mettre dans un programme?
- Concept de base: variable, type, classe, objet, héritage (OO), ...
- Utilisation des packages existants
- Principe de gestion d'interface graphique
- Principe de télécommunication en Java