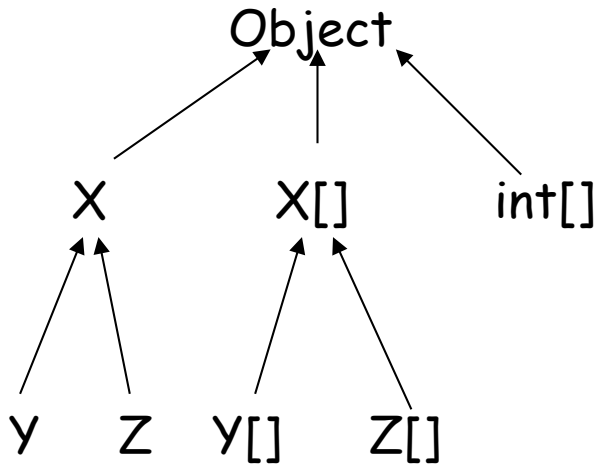
# Cours 8
# (18 novembre 2011)

exceptions

# Tableau et héritage




- `Y[] yA=new Y[3];`
- `X[] xA=yA; //ok`
- `xA[0]=new Y();`
- `xA[1]=new X(); //non`
- `xA[1]=new Z(); //non`

# Noms

- il ya 6 espaces de noms
  - package
  - type
  - champs
  - méthode
  - variable locale
  - étiquette

# Noms!?

```
package divers;
class divers{
    divers divers(divers divers){
        divers:
            for(;;){
            if (divers.divers(divers)==divers)
                break divers;
            }
            return divers;
    }
}
```

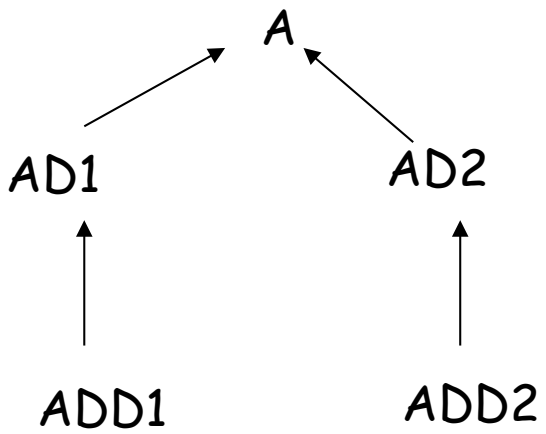
# Trouver la bonne méthode

- Il faut pouvoir déterminer une seule méthode à partir d'une invocation avec des paramètres
- problèmes: héritage et surcharge
  - (en plus des problèmes liés à la généricité)
- principe trouver la méthode "la plus spécifique"

# Règles

1. *déterminer dans quelle classe chercher la méthode (uniquement par le nom)*
2. trouver les méthodes dans la classe qui peuvent s'appliquer
    1. sans "boxing" sans nombre variable d'arguments
    2. avec boxing
    3. avec un nombre variable d'arguments
3. si une méthode a des types de paramètres qui peuvent être affectés à une autre des méthodes de l'ensemble -> la supprimer
4. s'il ne reste qu'une méthode c'est elle (sinon ambiguïté sauf s'il s'agit de méthodes abstraites)

# Exemple




```
void f(A a,AD2 ad2)//un
void f(AD1 ad1,A a)//deux
void f(ADD1 add1, AD2 s)//trois
void f(A … a)//quatre

f(Aref, AD2ref);//a
f(ADD1ref, Aref);//b
f(ADD1ref, ADD2ref);//c
f(AD1ref, AD2ref);//d
f(AD2ref, AD1ref);//e
```

# Exemple (suite)

- (a) correspond exactement à (un)
- (b) correspond à (deux)
- (c) peut correspondre aux trois premiers mais (un) est moins spécifique que (trois) idem entre (un) et (trois) d'où résultat (trois)
- (d) (un) et (deux) ne peuvent s'éliminer
- (e) uniquement (quatre)

# Chapitre VI
# Exceptions

# VI) Exceptions

1. Principes généraux
2. Déclarations de throws
3. try, catch et finally
4. Transfert d'information: chainage, pile
5. Assertions

# Exceptions et assertions

- principe:
  - traitement des "erreurs"
    - quand une exception est lancée:
      - rupture de l'exécution séquentielle
      - "dépiler" les méthodes et les blocs
      - jusqu'à un traite exception adapté
  - erreur:
    - rupture du contrat:
      - précondition violée

# Exceptions

- checked ou unchecked
  - checked: cas exceptionnel, mais dont l'occurrence est prévue et peut être traitée (exemple: valeur en dehors des conditions de la précondition, ...)
    - Une méthode qui peut lancer une checked exception doit le déclarer
  - unchecked: il n'y a rien à faire, (exemple une erreur interne de la JVM) ou une erreur à l'exécution (dépassement de tableau)
    - Une méthode qui peut lancer une unchecked exception ne doit pas le déclarer

# Exceptions

- Une exception est un objet d'une classe dérivée de Throwable (mais, en fait, en général de Exception)
- Le mécanisme est le même que pour tout objets:
  - on peut définir des sous-classes
  - des constructeurs
  - redéfinir des méthodes
  - ajouter des méthodes

# Exceptions et traite-exceptions

- Un traite exception déclare dans son entête un paramètre
- Le type du paramètre détermine si le traite-exception correspond à l'exception
  - même mécanisme que pour les méthodes et l'héritage

# Throw

- Certaines exceptions et errors sont lancées par la JVM
- L'utilisateur peut définir ses propres exceptions et les lancer lui-même:

  `throw expression;`

  l'expression doit s'évaluer comme une valeur ou une variable qui peut être affectée à Throwable

# Environnement

- Par définition une exception va transférer le contrôle vers un autre contexte
  - le contexte dans lequel l'exception est traitée est différent du contexte dans lequel elle est lancée
  - l'exception elle-même peut permettre de passer de l'information par son instanciation
  - l'état de la pile au moment de l'exception est aussi transmis
    public StackTraceElement[] **getStackTrace**() et
    public void **printStackTrace**()

# Hiérarchie:

- java.lang.Throwable (implements java.io.Serializable)
  - java.lang.Error
    - java.lang.AssertionError
    - java.lang.LinkageError
    - java.lang.ThreadDeath
    - java.lang.VirtualMachineError
      - exemple:java.lang.StackOverflowError
  - java.lang.Exception
    - java.lang.ClassNotFoundException
    - java.lang.CloneNotSupportedException
    - java.lang.IllegalAccessException
    - java.lang.InstantiationException
    - java.lang.InterruptedException
    - java.lang.NoSuchFieldException
    - java.lang.NoSuchMethodException
    - java.lang.RuntimeException
      - exemple: java.lang.IndexOutOfBoundsException

# Hiérarchie

- Throwable:
  - la super classe des erreurs et des exceptions
  - Error : unchecked
  - Exception :checked sauf RuntimeException

# Exemple

```
public class MonException extends Exception{
    public final String nom;
    public MonException(String st) {
        super("le nombre "+st+" ne figure pas");
        nom=st;
    }
}
```

# Exemple (suite)

```
class Essai{
    static String[] tab={"zéro","un","deux","trois","quatre"};
    static int chercher(String st ) throws MonException{
        for(int i=0;i<tab.length;i++)
            if (tab[i].equals(st))return i;
        throw new MonException(st);
    }

    public static void main(String st[]){
        try{
            chercher("zwei");
        }catch(Exception e){
            System.out.println(e);
        }

    }
}
```

# Résultat

- Donnera:

```
exceptions.MonException: le nombre zwei ne figure pas
```

- e.printStackTrace(); dans le try bloc donnera:

```
exceptions.MonException: le nombre zwei ne figure pas
        at exceptions.Essai.chercher(MonException.java:29)
        at exceptions.Essai.main(MonException.java:34)
```

# Throws

- principe:
  - toute méthode qui peut générer directement ou indirectement une (checked) exception doit le déclarer par une clause "throws" dans l'entête de la méthode.
    - (les initialiseurs statiques ne peuvent donc pas générer d'exceptions)
  - La vérification a lieu à la compilation

# Clause throws

- Une méthode qui appelle une méthode qui peut lancer une exception peut
  - attraper (catch) cette exception dans un try bloc englobant la méthode qui peut lancer cette exception
  - attraper cette exception et la transformer en une exception déclarée dans la clause throws de la méthode
  - déclarer cette exception dans la clause throws de sa déclaration

# Clause throws et héritage

- Si une classe dérivée redéfinit (ou implémente) une méthode la clause throws de la méthode redéfinie doit être compatible avec celle d'origine
  - compatible = les exceptions de la clause throws sont dérivées de celles de la méthode d'origine
  - pourquoi?

# try, catch, finally

- On attrape les exceptions dans des try-bloc:

```
try{
   instructions
}catch(exception-type1 id1){
     instructions
} catch(exception-type2 id2){
     ...
}finally{
   instructions
}
```

# Principe:

- le corps du try est exécuté jusqu'à ce qu'il termine ou qu'une exception est lancée
- Si une exception est lancée les clauses "catch" sont examinées dans l'ordre
  - la première dont le type peut correspondre à l'exception est choisie et son code exécuté
  - si aucun catch ne peut correspondre l'exception est propagée
  - si une clause finally figure son code est ensuite exécuté (toujours avec ou sans exception)

# Exemple

```
class A extends Exception{
}
class B extends A{
}
class essai{
    public static void main(String[] st){
        try{
            throw new B();
        }catch (A a){
            System.out.println(a);
//        }catch (B b){
//            System.out.println(b);
        }finally{
            System.out.println("finally..");
        }
    }
}
```

# finally

```
public boolean rechercher(String fichier,
      String mot) throws StreamException{
        Stream input=null;
        try{
            input=new Stream(fichier);
            while(!input.eof())
                if(input.next().equals(mot))
                    return true;
            return false;
        }finally{
            if (input != null)
                input.close();
        }
    }
```

# Chaînage d'exceptions

- Une exception peut être causée par une autre.
- il peut être utile dans ce cas de transmettre la cause de l'exception
  - méthode:

public Throwable **initCause**(Throwable cause)

# Transmission d'information

- en définissant une extension de la classe et en définissant des constructeurs
- par défaut on a les constructeurs

  public **Throwable**()

  public **Throwable**(String message)

  public **Throwable**(String message, Throwable cause)

# Transmission d'information

- On peut récupérer ces informations:

```
public String getMessage()
public Throwable getCause()
```

- On peut obtenir l'état de la pile:

```
public void printStackTrace()
public StackTraceElement[]
   getStackTrace()
```

# Exemple

```
class X extends Exception{
    public X(){}
    public X(String details){
        super(details);
    }
    public X(Throwable e){
        super(e);
    }
    public X(String details, Throwable e){
        super(details,e);
    }
}
```

# Suite

```
try{
            throw new A();
        }catch (A a){
            try {
                throw new X(a);
            } catch (X ex) {
                ex.printStackTrace();
            }
        }
    }
-----
X: A
        at essai.main(Finally.java:61)
Caused by: A
        at essai.main(Finally.java:58)
```

# Remarque

- à la place de:

```
throw new X(a);
```

- on pourrait mettre

```
throw (X) new X().initCause(a);
```

(pourquoi le cast (X) est nécessaire?)

# Assertions

- Une autre façon de garantir le contrat est de définir des assertions qui vérifient les invariants (ou les préconditions)
- Si l'assertion n'est pas vérifiée une `AssertionError` est lancée
- (une option de compilation permet de vérifier ou non les assertions)

# Assertions

- Syntaxe:

  `assert` expr [: detail];

  - expr est une expression `boolean`
  - detail est optionnel et sera passé au constructeur de `AssertionError`

    (une string ou un `Throwable`)

- Exemple:

```
assert i!=0 : "i ="+i+" i devrait être non nul";
```

# Assertions

- par défaut les assertions ne sont pas évaluées
- pour les évaluer:
  - -enableassertions:nom_du_package
  - -disableassertions:nom_du_package

  avec en argument le ou les paquetages concernés.