

Programmation Orientée Objet - Licence TIS CM6/9

Lancelot PECQUET

Lancelot.Pecquet@math.univ-poitiers.fr



Poitiers, le 06/02/2006

- 1 Transtypage
- 2 Égalité
- 3 Comparabilité
- 4 Clonage

Rappel sur la séance précédente

La fois précédente, nous avons vu :

- 1 interruptions
- 2 paquetages
- 3 javadoc
- 4 collections et itérateurs
- 5 quelques nouveautés de Java 1.5

Aujourd'hui, nous voyons :

- 1 transtypage
- 2 égalité
- 3 comparabilité
- 4 clonage

Définition du transtypage (cast)

Définition du cast

Soient T_1 et T_2 deux types. Un **transtypage (cast)** de T_1 vers T_2 est une application :

$$\varphi : \quad T_1 \xrightarrow{\text{cast}} T_2$$

$$x_1 \mapsto x_2 = \varphi(x_1)$$

qui à tout objet $x_1 : T_1$ fait correspondre un objet $x_2 : T_2$.

Syntaxe

- si pas de confusion, on note $(T_2)x_1$ au lieu de $\varphi(x_1)$
- invocation de méthodes prioritaire sur le cast :
 - $(T)M.f()$ appel de f depuis M puis cast du résultat en T
 - $((T)M).f()$: transtypage de M en T puis invocation de f

Définition d'un *upcast* et d'un *downcast*

Définition d'un *upcast* et d'un *downcast*

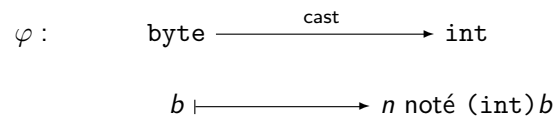
- types : arbre orienté des feuilles vers la racine
- fonctions de transtypages φ injectives (sauf pour `float` et `double`, bien que mathématiquement, \mathbb{Z} s'injecte dans \mathbb{R})
- remonte dans l'arbre (on applique un φ) : *upcast*
- descend dans l'arbre (on applique un φ^{-1}) : *downcast*
- si φ est non surjective sur T_2 :
 - soit on prolonge φ^{-1} : choix non canonique (types primitifs)
 - soit on interdit φ^{-1} : `ClassCastException`

Transtypage Java

- transtypage des types primitifs
- transtypage des objets dans le cadre de l'héritage

Upcast des types primitifs

- considérons l'application :



- la syntaxe de base pour transtyper `b` de `byte` en `int` est :

```
1 byte b = 53; // b = 53 est un entier 8bit...
2 int n = (int)b; // ...qui s'identifie naturellement a l'entier 32bit n = 53
```

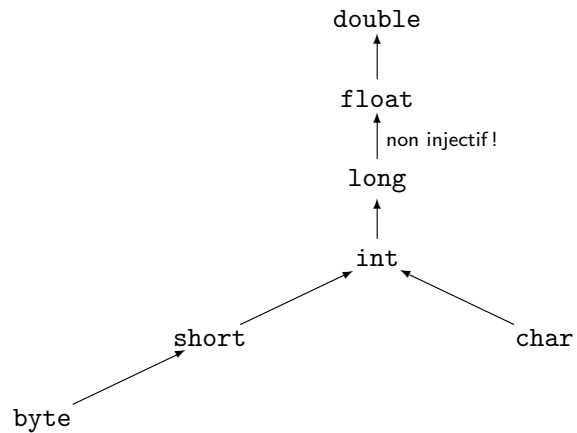
Identification naturelle : *upcast* automatique

- φ est une injection canonique (naturelle)
- on peut toujours naturellement identifier le byte `b` à l'`int` `n`
- *upcast* automatique :

```
1 byte b = 53;  
2 int n = b; // identification naturelle = upcast automatique sans (int)
```

Upcasts automatiques pendant l'arithmétique

Ceux-ci ont lieu avant une opération, si besoin :



Downcast des types primitifs

- φ n'est pas surjective (on a $|\text{byte}| = 2^8 < |\text{int}| = 2^{32}$)
- φ^{-1} (*downcast*) n'est pas naturel
- il peut y avoir perte d'information
- il faut être sûr de ce qu'on fait : pas de *downcast* automatique
- exemple de situation problématique :

```
1 int n = 1000; // en binaire n: 0000 0000 0000 0000 0000 0011 1110 1000
2 byte b = (byte)n; // b = -24 car, en tronquant n a 8bits, b: 1110 1000
```

Ariane V : 10 ans de travail, développement : 10MM€

- paramètres moteur stockés sur des `double` (64 bits)
- pour les calculs : *downcast* en `short` (16 bits)
- 1996 : premier vol pour placer en orbite deux satellites
- l'un des paramètres dépasse la valeur du plus grand `short` et ne peut être *downcasté*
- il y a *overflow* en cascade et plantage généralisé
- la fusée se désintègre après 40s de vol
- coût : 500M€
- explication :
 - réutilisation du code d'Ariane IV pour Ariane V
 - pas d'adaptations
 - Ariane V est plus puissante qu'Ariane IV...

Transtypage des objets

- polymorphisme d'héritage = *upcast* automatique :

```
1 class pixel extends point {...}
2 ...
3 point A = new point(320,200);
4 pixel p = new pixel(A,noir);
5 p.set_x(10); // Polymorphisme: pas besoin d'ecrire (point)p.set_x()
```

- on ne peut downcaster que les "héritiers masqués" :

```
1 point q = (point)p; // Downcast explicite (couleur masquee mais conservee)
2 ((pixel)q).set_c(blanc); // OK car q est un point qui "provient" d'un pixel
3 ((pixel)A).set_c(blanc); // ClassCastException: car A n'est pas un pixel!
```

Transtypage des tableaux

- on peut faire :

```
1 point[] nuage = new point[] {p,A}; // p est vu comme un point ici
```

- pour modifier la couleur de l'élément d'indice 0 de nuage, il faut le *downcaster* en pixel :

```
1 ((pixel)nuage[0]).set_c(blanc); // downcast explicite obligatoire: ok
2 ((pixel)nuage[1]).set_c(blanc); // ClassCastException: ce n'est pas un pixel!
```

- en fait :

```
1 Object o = new int[5]; // OK
2 class B extends A{};
3 A[] S;
4 B[] T;
5 S=T; // oui car il a upcast automatique lors du S[i]=T[i]
6 T=S; // non: downcast explicite necessaire
```

Savoir downcaster avec instanceof

- on peut avoir un problème de downcast à l'exécution (ClassCastException) :

```
1 Object x = (Object) (new Integer(1));  
2 String y = (String) x; // can't cast 'java/lang/Integer' to 'java/lang/String'
```

- pour le résoudre, avant de downcaster, on teste (pour les types non-primitifs) :

```
1 if (x instanceof ...) { ... }
```

Quiz instanceof

```
System.out.println("foo" instanceof String);
```

- true

```
System.out.println("foo" instanceof Object);
```

- true

```
System.out.println(1 instanceof Object);
```

- ne compile pas

```
System.out.println(1 instanceof int);
```

- ne compile pas

```
System.out.println(new Integer(1) instanceof Integer);
```

- true

Quiz instanceof (suite)

```
System.out.println((Object) "foo" instanceof String);
```

- true

```
System.out.println((Object) (new Integer(1)) instanceof String);
```

- false

```
System.out.println(new int[]{1} instanceof int[]);
```

- true

```
System.out.println(new int[]{1} instanceof Object);
```

- true

```
System.out.println(null instanceof Object);
```

- false

Questions ?



La méthode equals()

- l'opérateur == désigne l'égalité bit-à-bit :
 - des valeurs pour les type primitif
 - des références pour les types non-primitifs
- la sémantique du equals() hérité d'Object est ==
- autre sémantique \implies redefinition de equals()
- equals() est utilisée par toutes les bibliothèques standard

Exemple de redefinition la méthode equals()

Dans point.java

```
1 class point{...
2   public boolean equals(Object q){ // Noter que le parametre est de type Object
3     // Telle qu'elle, la ligne suivante peut lever une ClassCastException:
4     return ((get_x() == ((point)q).get_x()) && (get_y() == ((point)q).get_y()));
5   }
6   ...}
```

Dans Main.java

```
1 point p = new point(3,4);
2 point q = new point(3,4);
3 System.out.println(p == q); // false
4 // Noter que l'upcast de q vers Object est transparent:
5 System.out.println(p.equals(q)); // true
6 System.out.println(p.equals("toto")); // ClassCastException
```

Comment éviter un ClassCastException ?

Dans point.java

```
1 class point{...
2   public boolean equals(Object g){
3     boolean b;
4     if (g instanceof point){
5       b = ((get_x() == ((point)g).get_x()) && (get_y() == ((point)g).get_y()));
6     }else{
7       b = false;
8     }
9     return b;
10  }
11  ...}
```

Problème

Difficultés à gérer equals() avec l'héritage.

Questions ?



Interface Comparable et méthode compareTo()

- relation d'ordre = interface Comparable
- implémentation de `public int compareTo(Object o) :`

```
1 class point implements Comparable{...
2 // Définissons l'ordre lexicographique
3 public int compareTo(Object o){ // Noter que le parametre est un Object
4     point q = (point)o; // Cette version peut lever une ClassCastException
5     if(get.x() < q.get.x()){return -1;}
6     else if (get.x() > q.get.x()){return 1;}
7     else{
8         if(get.y() < q.get.y()){return -1;}
9         else if(get.y() > q.get.y()){return 1;}
10        else{return 0;}
11    }
12 ...}
```

- Object n'est pas Comparable.

Exemple de Comparable : la classe String

L'ordre sur String est lexicographique :

```
1 String[] S = new String[]{"foo", "bar", "gee"};
2
3 // La ligne suivante renvoie 4>0 (nb de lettres entre c et f):
4 System.out.println("foo".compareTo("bar"));
5
6 System.out.println("foo".compareTo("foo")); // renvoie 0
7
8 // La ligne suivante renvoie -4<0 (nb de lettres entre b et f):
9 System.out.println("bar".compareTo("foo"));
10
11 // La methode compareTo() est standard:
12 Array.sort(S);
```

Questions ?



Principe du clonage

- duplication des instances
- les copies sont indépendantes
- implementation de l'interface Cloneable et de la méthode `protected Object clone() throws CloneNotSupportedException`

Exemple simple de clonage

Fichier point.java

```
1 class point implements Cloneable{...
2
3 public Object clone(){
4     point q = null;
5     try{q = (point)super.clone();} // il faut toujours commencer par la
6     catch(CloneNotSupportedException e){}
7     return q;
8 }
9 ...}
```

Exemple simple de clonage

Fichier Main.java

```
1 point p = new point(3,2);
2 point q = p.clone();
3 p.setX(10); // Cela n'affecte pas le clone
4 System.out.println(p);
5 System.out.println(q);
```

Sortie (console)

```
1 (10,2)
2 (3,2)
```

La méthode clone() d'Object

Son comportement subtil dépend du type des valeurs en jeu :

- ① primitif (int, char,...);
- ② non-primitif :
 - ① instance de classe (String, Integer...);
 - ② tableau d'éléments de type primitif (int []...).
 - ③ tableau d'éléments de type non-primitif (String [], int [] []...).

La méthode clone() d'Object

- le profil d'un clone() est le suivant :

```
1 class K implements Cloneable{...
2   public Object clone() {
3     K z = null;
4     try{
5       z = (K) super.clone();
6     }
7   } catch(CloneNotSupportedException e){ // 'inutile' mais obligatoire
8     return z;
9   }
10  ...}
```

- à la ligne 6, l'instance z a été allouée par le clone() d'Object à la ligne précédente et pour chacun des champs c de la classe K, il y a plusieurs cas selon le type de c.

Clonage des champs primitifs et tableaux

Le `clone()` d'`Object` a effectué `z.c = c` i.e. `z.c` est une copie bit-à-bit de `c` i.e. `z.c == c` est vrai.

- 1 si `c` est un champ primitif alors `z.c` et `c` sont donc deux valeurs égales mais distinctes en mémoire ;
- 2 si `c` est un tableau de type `T[]` alors `z.c` et `c` désignent la même référence, ce qui n'est pas ce qu'on veut dans un clonage. L'ajout de l'instruction `z.c = (T[])c.clone()` à la ligne 6 équivaut au code suivant :

```
1 T[] c.clone = new T[c.length];  
2 for(int i=0; i<c.length; i++){ c.clone[i] = c[i];}  
3 z.c = c.clone;
```

Clonage des champs primitifs et tableaux (suite)

Ainsi `z.c[i]` est une copie bit-à-bit de `c[i]`, de telle sorte que l'expression `z.c[i] == c[i]` est vraie, pour tout `i` entre 0 et `c.length`.

- 1 si `T` est un type primitif, c'est suffisant pour que le tableau ait été cloné en profondeur ;
- 2 sinon, `z.c[i]` et `c[i]` désignent la même référence. Restent deux cas :
 - 1 si `T` est immuable, alors on peut considérer le clonage comme satisfaisant car on ne pourra jamais modifier l'objet `X` référencé par `c[i]` ;
 - 2 sinon, afin de dupliquer effectivement les `c[i]`, il convient d'ajouter, à la ligne 7, le code suivant :

```
1 for(int i=0; i<c.length; i++){ z.c[i] = (K)c[i].clone();}
```

Clonage des autres champs

`c` est l'instance d'une classe `C` et le `clone()` d'`Object` a effectué récursivement l'opération `z.c = (C)c.clone()`. Attention, ici `c` est vu comme un `Object` et, même si `clone()` a été redéfini dans la classe `C`, c'est celui d'`Object` qui s'appliquera (donc pas forcément comme on le souhaiterait).

Exemple de clonage

```
1 class K implements Cloneable{...
2   int c1; // type primitif
3   int[] c2; // tableau unidimensionnel d'elements de type primitif
4   int[][] c3; // tableau bidimensionnel d'elements de type primitif
5   C c4; // instance de classe
6   C[] c5; // tableau unidimensionnel d'instances de classe
7   C[][] c6; // tableau bidimensionnel d'instances de classe
8   ...
```

Exemple de clonage (suite)

```
1 public Object clone(){
2     K z = null;
3     try{
4         z = (K) super.clone();
5
6         /* a ce stade, c1 a été clone convenablement. C'est aussi le cas
7            pour c4 si C est immuable ou ne contient aucun tableau dans
8            ses champs (et récursivement pour les champs de ses champs,
9            etc.) */
10
11        z.c2 = (int[])c2.clone(); // suffit à cloner c2
12
13        z.c3 = (int[][]c3.clone();
14        for(int i=0;i<c3.length;i++){z.c3[i] = (int[])c3[i].clone();}
15
16        // clone c4 à coup sûr si C implémente convenablement clone():
17        z.c4 = (C)c4.clone();
```

Exemple de clonage (fin)

```
1
2 // suffit à cloner c5 si C est immuable:
3 z.c5 = (C[])c5.clone();
4
5 // nécessaire pour un C général:
6 for(int i=0;i<c5.length;i++){z.c5[i] = (C)c5[i].clone();}
7
8 z.c6 = (C[][]c6.clone();
9 for(int i=0; i<c6.length; i++){
10     for(int j=0; j<c6[0].length; j++){
11         z.c6[i][j] = (C)c6[i][j].clone();
12     }
13 }
14 }catch(CloneNotSupportedException e){}
15 return z;
16 }
17 ...}
```

Exemple de clonage

Fichier telephone.java

```
1 class telephone implements Cloneable{
2   public static final int NB_CHIFFRES = 10;
3   private int[] num;
4   public final int getNum(int i){return num[i];}
5   public final void setNum(int i, int d){num[i]=d;}
6   private final void setNum(int[] num){this.num=num;}
7   private final int[] getNum(){return num;}
8
9   public telephone(){num=new int[NB_CHIFFRES];}
10  public telephone(String s){
11    this();
12    for(int i=0;i<NB_CHIFFRES;i++){
13      setNum(i,Integer.parseInt(s.substring(i,i+1)));
14    }
15  }
```

Exemple de clonage

Fichier telephone.java (suite)

```
1 class telephone implements Cloneable{...
2
3   public String toString(){
4     StringBuffer s = new StringBuffer();
5     for(int i=0;i<NB_CHIFFRES;i+=2){
6       s.append(Integer.toString(getNum(i)) + getNum(i+1) + "-");
7     }
8     return s.toString();
9   }
10
11  ...}
```

Exemple de clonage

Fichier telephone.java (suite)

```
1 class telephone implements Cloneable{...
2
3 public Object clone(){
4     telephone T = null;
5     try{
6         T = (telephone)super.clone(); // la ref. num est commune entre T et this
7         setNum((int[])getNum().clone()); // on clone num (int primitif => copie OK)
8     }catch(CloneNotSupportedException e){}
9     return T;
10 }
11 public static void test(){
12     // le setNum((int[])num.clone()) de clone() est necessaire, (essayer sans)
13     telephone T = new telephone("0132245576");
14     telephone S = (telephone)T.clone();
15     S.setNum(0,9);
16     System.out.println(T);
17     System.out.println(S);
18 }
19 }
```

Exemple de clonage

Fichier personne.java

```
1 class personne implements Cloneable{
2     private String nom, prenom;
3     private int age;
4     private telephone domtel;
5     private telephone[] nums; // bureau, mobile
6     public final void setNom(String nom){this.nom=nom;}
7     public final void setPrenom(String prenom){this.prenom=prenom;}
8     public final void setAge(int age){this.age=age;}
9     public final void setTel(int i, telephone T){nums[i]=T;}
10    public final void setDomTel(telephone T){domtel=T;}
11    public final String getNom(){return nom;}
12    public final String getPrenom(){return prenom;}
13    public final int getAge(){return age;}
14    public final telephone getDomTel(){return domtel;}
15    public final telephone getTel(int i){return nums[i];}
16    private final telephone[] getNums(){return nums;}
17    private final void setNums(telephone[] nums){this.nums=nums;}
18    ...}
19 }
```

Exemple de clonage

Fichier personne.java (suite)

```
1 class personne implements Cloneable{...
2
3     public personne(String nom, String prenom, int age){
4         setNom(nom); setPrenom(prenom); setAge(age);
5         domTel = new telephone();
6         nums = new telephone[2];
7         for(int i=0; i<nums.length; i++){setTel(i, new telephone());}
8     }
9
10    public String toString(){
11        return getNom() + "-" + getPrenom() + "-" + getAge() + "-ans"
12            + "\ntel._dom:_" + getDomTel()
13            + "\ntel._bur:_" + getTel(0)
14            + "\ntel._mob:_" + getTel(1);
15    }
16    ...}
```

Exemple de clonage

Fichier personne.java (suite)

```
1 class personne implements Cloneable{...
2
3     public Object clone(){
4         personne p = null;
5         try{
6             // clone bien nom, prenom, age:
7             p = (personne)super.clone();
8
9             // il faut cloner explicitement:
10            p.setDomTel((telephone)getDomTel().clone());
11
12            // clonage superficiel des autres telephones:
13            setNums((telephone[])getNums().clone());
14
15            // clonage en profondeur:
16            for(int i=0; i<nums.length; i++){
17                setTel(i, (telephone)getTel(i).clone());
18            }
19        }catch(CloneNotSupportedException e){}
20        return p;
21    }
22    ...}
```

Exemple de clonage

Fichier personne.java (suite)

```
1 class personne implements Cloneable{...
2
3     public static void test(){
4         // le for...setTel(1,(telephone)getTel(i).clone()) de clone()
5         // est necessaire (essayer sans)
6         personne john = new personne("John","Doe",30);
7         john.setDomTel(new telephone("0143568766"));
8         john.setTel(0,new telephone("0299872265"));
9         john.setTel(1,new telephone("0622997365"));
10        personne john2 = (personne)john.clone();
11        john2.setAge(43); // le clone() d'Object a suffi (primitif)
12        john2.setNom("Nob"); // le clone() d'Object a suffi (objet immuable)
13        john2.getTel(0).setNum(0,9); // le clone() d'Object est insuffisant
14        john2.getDomTel().setNum(0,7); // le clone() d'Object est insuffisant
15        System.out.println(john);
16        System.out.println(john2);
17    }
18 }
```

Exemple de clonage

Fichier Main.java

```
1 class Main{
2     public static void main(String[] args){telephone.test(); personne.test();}
3 }
```

sortie (console)

```
1 01 32 24 55 76
2 91 32 24 55 76
3 John Doe (30 ans)
4 tel. dom: 01 43 56 87 66
5 tel. bur: 02 99 87 22 65
6 tel. mob: 06 22 99 73 65
7 Nob Doe (43 ans)
8 tel. dom: 71 43 56 87 66
9 tel. bur: 92 99 87 22 65
10 tel. mob: 06 22 99 73 65
```

Questions ?



Conclusion

Aujourd'hui, nous avons vu :

- 1 Transtypage
- 2 Égalité
- 3 Comparabilité
- 4 Clonage

La séance prochaine, nous verrons :

- 1 présentation du projet
- 2 conception OO
- 3 UML
- 4 design patterns