Clonage des objets

- *clone()* définie dans la classe **Object**
- **Object.clone()** fait par défaut une **copie superficielle (shallow copy)** si l'objet implémente l'interface **Cloneable :**
    - **pour les types de base, les valeurs sont recopiées**
    - **pour les types objets, seules les références sont recopiées et pas les objets référencés.**
- *clone()* a une visibilité **protected dans la classe Object**
- *clone()* peut lever une exception **CloneNotSupportedException**
- L'interface **Cloneable est une interface de marquage**
  ==> la méthode *clone()* n'y figure pas!
  L'interface **Cloneable** ne contient ni constante ni prototype de méthode, elle ne sert que pour savoir si il est prévu ou non qu'un objet soit cloné.

COPIE SUPERFICIELLE / SHALLOW COPY

La méthode clone duplique tous les attributs d'une classe mais pas le "contenu" des attributs de type référence.

Si une classe A contient :

- un attribut entier n
- un attribut t référençant un tableau (ou d'un objet)

la méthode clone appliquée à une instance a de A construit une nouvelle instance aCopie de A avec :

- un attribut entier n qui a la même valeur que l'attribut n de l'instance a
- une référence t qui référence le même tableau que l'attribut t de l'instance a.

Pour que les instances d'une classe contenant des références puissent être dupliquées, il faut ou bien que la classe redéfinisse la méthode clone, ou bien qu'elle prévoit sa propre méthode de duplication.

De toute manière, la méthode **clone** de la classe **Object** étant protégée (i.e. déclarée **protected**), il faut, pour qu'une instance d'une classe A extérieure au paquetage **java.lang** puisse invoquer cette méthode, redéfinir la méthode clone dans la classe A.

Par ailleurs, pour pouvoir utiliser la méthode clone de la classe Object sur les instances d'une classe, il faut que cette dernière **implémente l'interface Cloneable du paquetage java.lang**. En effet, la méthode clone de la classe Object vérifie que l'objet qui l'invoque est d'une classe implémentant cette interface et si ce n'est pas le cas, lance une **exception de la classe CloneNotSupportedException.**

Si une classe redéfinit la méthode clone, il est pertinent qu'elle implémente l'interface Cloneable.

I Duplication d'un objet avec la méthode clone de Object

La méthode **clone de Object** lance systématiquement l'exception **CloneNotSupportedException** si la classe n'implémente pas l'interface **Cloneable**.

```
package clone;

public class MaClasse1 implements Cloneable {
	private int i = 0;

	public static void main(String [] argv) {
		MaClasse1 o1 = new MaClasse1();

		try {
			MaClasse1 o2 = (MaClasse1) o1.clone(); // o2 est un clone de o1
			o2.i = 1; // o1.1 n'est pas modifié
		} catch(CloneNotSupportedException e) {
			System.out.println(e);
		}

		System.out.println(o1.i); // o1.i n'a pas changé
	}
}
```

II Clonage d'un objet avec des références

1) La méthode clone de objet ne peut pas être utilisée pour cloner des objets ayant des références :

```
package clone;

public class MaClasse2 implements Cloneable {
	private int [] tableau = new int[5];

	public static void main(String [] argv) {
		MaClasse2 o1 = new MaClasse2();
		o1.tableau[0] = -1;
		System.out.println(o1.tableau[0]); // -1

		try {
			MaClasse2 o2 = (MaClasse2) o1.clone(); // o2 est une clone de o1
			o2.tableau[0] = 0; // le tableau est commun!
		} catch( CloneNotSupportedException e) {
			System.out.println(e);
		}

		System.out.println(o1.tableau[0]); // 0 changement !!
	}
}
```

2) Pour cloner un objet ayant des références, il faut redéfinir la méthode clone :

```
package clone;

public class MaClasse3 implements Cloneable {
	private int [] tableau = new int[5]; // méthode clone définie pour des tableaux

	@Override
	public Object clone() throws CloneNotSupportedException {
		MaClasse3 obj = (MaClasse3) super.clone();
		obj.tableau = (int []) tableau.clone(); // clone existe!
		return obj;
	}

	public static void main(String [] argv) {
		MaClasse3 o1 = new MaClasse3();
		o1.tableau[ 0 ] = -1;
		System.out.println(o1.tableau[0]);

		try {
			MaClasse3 o2 = (MaClasse3) o1.clone(); // clonage OK
			o2.tableau[0] = 0; // seul o2 est modifé
		} catch(CloneNotSupportedException e) {
			System.out.println(e);
		}

		System.out.println(o1.tableau[0]); // -1 : pas de changement
	}
}
```

```java
import java.util.Date;

public class MaClasse4 implements Cloneable {
	private Date date = new Date();  // heure courante du système

	@Override
	public Object clone() throws CloneNotSupportedException {
		MaClasse4 obj = (MaClasse4)super.clone();
		obj.date = (Date) date.clone();
		return obj;
	}

	public static void main(String [] argv) {
		MaClasse4 o1 = new MaClasse4();
		System.out.println(o1.date);

		try {
			MaClasse4 o2 = (MaClasse4) o1.clone(); // clonage OK
			o2.date.setYear( 48 ); // seul o2 est modifié
		} catch(CloneNotSupportedException e) {
			System.out.println(e);
		}

		System.out.println(o1.date) ; // pas de changement
	}
}
```

III Interdire le clonage

- ne pas implémenter Cloneable ;

ou bien

- **définir clone en lançant systématiquement une exception.**

```
package clone;

public class MaClasse5 {

	@Override
	public Object clone() throws CloneNotSupportedException {
		throw new CloneNotSupportedException();
	}

	public static void main( String [] argv ){
		MaClasse5 o1 = new MaClasse5() ;

		try{
			MaClasse5 o2 = (MaClasse5)o1.clone() ;
		} catch( CloneNotSupportedException e ) {
			System.out.println(e) ;
		}
	}
}
```

IV Cloner un objet sans attribut contenant un objet mutable

```
package clone;

public class MyInteger implements Cloneable { // nécessaire pour Object.clone()
	private final int value;

	public MyInteger(int value) {
		this.value=value;
	}

	/**
	 * FACULTATIF puisque Copie superficielle suffisante
	 * la méthode clone de Object suffit
	 */
	/*
	@Override
	public MyInteger clone() throws CloneNotSupportedException {
		return (MyInteger)super.clone(); // shallow copy, cast MyInteger
	}
	*/

	public static void main(String[] args) {
		MyInteger i = new MyInteger(3);
		MyInteger j = null;
		try {
			j = (MyInteger) i.clone();
			// j = i.clone(); SI clone redéfinit
		} catch (CloneNotSupportedException e) {
			e.printStackTrace();
		}

		System.out.println(i == j);		// false
		System.out.println(i.equals(j));	// true
	}

	@Override
	public String toString() {
		return "MyInteger [value=" + value + "]";
	}

	@Override
	public int hashCode() {
		...
	}

	@Override
	public boolean equals(Object obj) {
		...
	}
}
```

IV Cloner un objet avec des attributs contenant des objets mutables

- appeler **clone()** sur l'objet mutable

```
package clone;
import java.util.Date;

public class Mutable {
	private Date date; // classe Date mutable

	public Mutable(Date date) {
		// pas de copie défensive ==> effet de bord possible
		this.date = date;
	}

	public Date getDate(){
		// pas de copie défensive ==> effet de bord possible
		return date;
	}

	@Override
	public Mutable clone() throws CloneNotSupportedException {
		Mutable mutable = (Mutable)super.clone(); // shallow copy
		mutable.date = (Date) this.date.clone(); // clone la date
mutable
		return mutable;
	}

	@Override
	public String toString() {
		return date.toString();
	}
}
```

```
package clone;

public class MyHolder implements Cloneable { // nécessaire pour Object.clone()
	private Mutable mutable;

	public MyHolder(Mutable mutable) {
		this.mutable=mutable;
	}

	@Override
	public MyHolder clone() throws CloneNotSupportedException {
		MyHolder holder = (MyHolder) super.clone(); // shallow copy
		holder.mutable = this.mutable.clone(); // clone le mutable
		return holder;
	}
}
```