Fiche Java

Types de base avec nombre de bits

boolean	true |false
char	16 bits Unicode
byte	entier signé 8 bits
short	entier signé 16 bits
int	entier signé 32 bits
long	entier signé 64 bits
float	flottant 32 bits
double	flottant 64 bits

Hashtable

Note :

- Si on put 2 objets identiques en tant qu'index, size() retourne 1.
- Si on put 2 objets différents ayant le même état sont considérés comme 2 clefs différentes. Pour empêcher cela, on peut redéfinir la méthode equals (Voir plus loin) et hashCode

```
Hashtable <String, Integer> tabHash = new Hashtable <String, Integer>();
tabHash.put("test", 1);
tabHash.put("5", 9);
tabHash.get("5"); // On y accède
```

```
public int hashCode() { return s.hashCode() ; } // s est un attribut de la classe
```

Constructeur appelant le précédent

```
public Main(String name, int age) {
	this.name = name;
	this.age = age;
}

public Main(int age) {
	this("Non renseigné", age); // On utilise le constructeur précédent
}
```

Héritage, polymorphisme, tests sur types et références

```
public class Vehicule {

	protected String marque;
	protected int vitesse;

	public Vehicule(String marque, int vitesse) {
```

```
		this.marque = marque;
		this.vitesse = vitesse;
	}

	public void afficher() {
		System.out.println("Marque : " + marque + " | Vitesse : " +
vitesse);
	}
}
```

```
public class Voiture extends Vehicule {

	private int nbPortes;

	public Voiture(String marque, int vitesse, int nbPortes) {
		super(marque, vitesse);
		this.nbPortes = nbPortes;
	}

	public void afficher() {
		System.out.println("Marque : " + marque + " | Vitesse : " + vitesse
+ " | Nombre de portes : " + nbPortes);
	}
}
```

```
public static void main(String[] args) {
	Voiture v = new Voiture("Mercedes", 220, 5);
	Vehicule v1 = new Voiture("BMW", 250, 3);
	Vehicule v2 = new Vehicule("Moto", 300);
	Vehicule v3 = (Vehicule) new Voiture("Audi", 300, 3);

	v.afficher();
	v1.afficher();
	v2.afficher();
	v3.afficher();

	v2 = v1;
	v2.afficher();
}
```

Sortie :

```
Marque : Mercedes | Vitesse : 220 | Nombre de portes : 5
Marque : BMW | Vitesse : 250 | Nombre de portes : 3
Marque : Moto | Vitesse : 300
Marque : Audi | Vitesse : 300 | Nombre de portes : 3
Marque : BMW | Vitesse : 250 | Nombre de portes : 3
```

Objets dérivés, objets de classe et cast

```
Voiture v = new Voiture("Mercedes", 220, 5);
Vehicule t = new Vehicule("Moto", 300);
v = (Voiture) t; // Ca fonctionne grace au cast (Fille vers Mère)
t = (Voiture) t; // On n'a pas d'erreur dans l'IDE mais à la compilation oui
v = t; // On a une erreur : "Type mismatch: cannot convert from Vehicule to
Voiture"
```

Méthodes abstraites

- Une classe contenant une méthode abstraite doit être déclarée abstraite,
- Une interface est une classe qui n'a **QUE** des méthodes abstraites,
- Les champs d'une interface sont par défaut static et final,
- Une interface peut étendre une autre interface,
- Les méthodes déclarées dans l'interface doivent être implémentées dans la classe utilisant l'interface

```
public interface Mouvement {

	public void avancer();
	public void reculer();
	public void garer();

}
```

```
public class Voiture extends Vehicule implements Mouvement {

	/** … **/

	public void avancer() {
		System.out.println("J'avance");
	}

	public void reculer() {
		System.out.println("Je recule");
	}

	public void garer() {
		System.out.println("Je me gare");
	}

}
```

Méthode toString()

```
public String toString() {
	String s = "Marque : " + marque + " | Vitesse : " + vitesse + " | Nombre de
portes : " + nbPortes;
	return s;
}

public void afficher() {
	System.out.println(this);
}
```

Méthode equals()

Sans redéfinir la méthode equals

```
Main o1 = new Main("Objet");
Main o2 = new Main("Objet");

if(o1.equals(o2))
	System.out.println("equals = true");
else
	System.out.println("equals = false");

// equals = false
```

On redéfinit la méthode

```
public boolean equals (Object obj) {
	if(obj != null && (obj instanceof Main))
		return s.equals(((Main)obj).s);
	else
		return false;
}

/** … **/

// equals = true
```

Méthode clone()

Pour qu'un objet puisse être cloné, il faut implémenter la classe de « Cloneable ».

```
Main o1 = new Main("Objet");
Main o2 = null;

try {
	o2 = (Main) o1.clone();
	o2.s = "test";
} catch (CloneNotSupportedException e) {
	System.out.println(e);
}

System.out.println("o1 : " + o1.s);  // o1 : Objet
System.out.println("o2 : " + o2.s);  // o2 : test
```

Note : Si l'objet a des références (Exemple un tableau), le tableau sera commun aux 2 objets. Pour empêcher cela, il faut redéfinir la méthode clone.

```
public Object clone() throws CloneNotSupportedException{
	Main obj = (Main)super.clone() ;
	obj.tableau = (int[]) tableau.clone() ;
	return obj ;
}
```

Les exceptions

La base : try, catch & finally

```
try {
	System.out.println(" => " + (1/0));
} catch (ClassCastException e) {
	e.printStackTrace();
}
System.out.println("On continue quand même");
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at fr.tests.Main.main(Main.java:15)
```

```
try {
	System.out.println(" => " + (1/0));
} catch (ClassCastException e) {
	e.printStackTrace();
}
finally {
	System.out.println("On continue quand même");
}
```

```
On continue quand même
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at fr.tests.Main.main(Main.java:17)
```

Exceptions personnalisées

```
public class CoordonnesException extends Exception {
	public CoordonnesException() {
		System.out.println("x doit être supérieur à 0 !");
	}
}
```

```
public static void main(String[] args) throws CoordonnesException {
	int x = 20, y = 0;

	if (x < 0)
		throw new CoordonnesException();
	else
		System.out.println("Coordonnées correctes");
}
```

- `throws CoordonnesException` : Indique qu'il peut y avoir danger et que s'il y a une erreur, elle sera traitée en tant qu'objet de la classe CoordonneesException.
- `throw new CoordonnesException()` : L'exception est levée (L'erreur apparait) car la classe est instanciée.
- Si ensuite on essaye de créer un objet Main, il faudra mettre son instanciation entre try {} catch { `CoordonnesException e` }