Les bases de Java

Présentation

Les caractéristiques essentielles de Java

Les concepteurs de Java (James Gosling et Bill Joy de chez Sun Microsystem) ont conçu Java comme un langage :

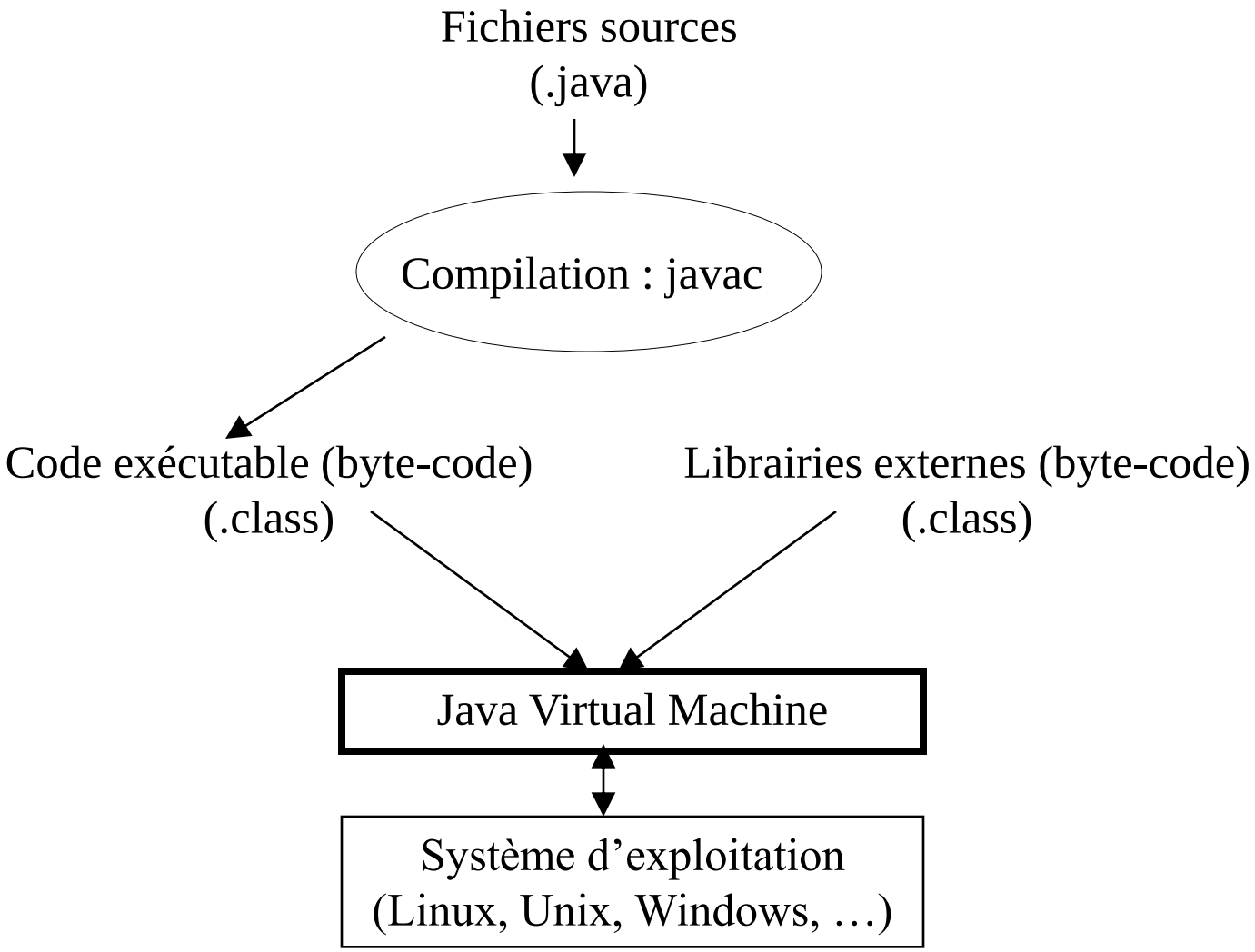
- indépendant de la machine et du système d'exploitation sous-jacent ;
- sûr pour pouvoir traverser un réseau ;
- puissant pour remplacer du code exécutable natif ;
- complet :
  - programmation réseau avancée (appel de méthode distante) ;
  - programmation d'interface utilisateur graphique ;
  - multi-thread ;
  - ...
- simple :
  - objet (syntaxe proche du C++) ;
  - où la gestion de mémoire n'est plus à la charge du programmeur.

Installer Java

- Java pour les programmeurs : Java Development Kit (JDK).
- Pour exécuter Java : Java Runtime environement.
  - http://www.oracle.com/technetwork/java/javase/downloads/index.html
  - Dernière version : 7
  - Version utilisée en entreprise : 6
  - L'aide en ligne : http://docs.oracle.com/javase/6/docs/api/
- Outils de développement :
  - Eclipse (http://www.eclipse.org/downloads/) : Eclipse IDE for Java developers.
  - NetBeans (http://fr.netbeans.org/).

Une machine virtuelle pour exécuter Java

Un programme Java s'exécute dans une machine virtuelle



Un exemple de programme

Exemple d'un programme

Fichier : Bonjour.java

```
public class Bonjour{          // classe Bonjour

      public static void main( String [] argv ) {
            System.out.println( "Bonjour" );
      }

}
```

- Compilation :
  - javac -d . Bonjour.java
- Lancer l'interpréteur Java :
  - java Bonjour

Les bases du langage

Les types de bases

- Java est un langage fortement typé.
- Grâce à la machine virtuelle Java, les types de base ont une taille fixe :

• boolean	->	true ou false ;
• char	->	caractère sur 16 bits Unicode ;
• byte	->	entier signé sur 8 bits ;
• short	->	entier signé sur 16 bits ;
• int	->	entier signé sur 32 bits ;
• long	->	entiers signé sur 64 bits ;
• float	->	flottant sur 32 bits ;
• double	->	flottant sur 64 bits.

Les tests

```
public class Bonjour{
	public static void main( String [] argv ) {

		int i = 0;
		if( i == 0 ){
			System.out.println( "i=" + i );
		}else{
			System.out.println( "i est non nul" );
		}

	}
}
```

```
public class Bonjour{
	public static void main( String [] argv ) {

		int i = 0;
		switch( i ){
			case 0 :
				System.out.println( "i=" + i );
			default :
				System.out.println( "i est non nul" );
		}

	}
}
```

Les boucles

```
public class Bonjour{
	public static void main( String [] argv ) {

		for( int i = 0; i<10; i++ ){
			System.out.println( "i=" + i );
		}

		int i=0;
		while( i<10 ){
			System.out.println( "i=" + i );
			i++;
		}

		i=0;
		do{
			System.out.println( "i=" + i );
			i++;
		}while( i<10 );

	}
}
```

Les tableaux

```
public class Bonjour{
	public static void main( String [] argv ) {

		int[] entiers = new int[ 10 ];
		entiers[ 0 ] = 0;
		System.out.println( "taille du tableau = " + entiers.length );

		char[] caractères = { ‘ a ’, ‘ b ’, ‘ c ’ };

		int taille = 10;
		int[] double = new int[ taille ];

		float matrice[][] = new float[ 10 ][ 10 ];
		matrice[ 0 ][ 0 ] = 0.0;

		for(int i=0; i<tab.length; i++){
			System.out.println(tab[i]);
		}

		for (int i : tab) {
			System.out.println(i);
		}

	}
}
```

Les fonctions

```
public class Bonjour{

	public String direBonjour(){
		return "Bonjour";
	}

	public static void main( String [] argv ) {

		Bonjour bonjour = new Bonjour();
		String s = bonjour.direBonjour();

	}
}
```

Surcharge des noms de méthodes

Plusieurs méthodes peuvent avoir le même nom.
Elles doivent se différencier par la liste de leurs arguments :

```
void methode( int i, int j ) {
        ...
}
void methode( int i, char j ) {
        ...
}
```

- Le choix de la méthode à appeler se fait au moment de la compilation ;
- C'est celle qui correspond le mieux à l'appel qui est choisie.

Surcharge des noms de méthodes

Le type de retour d'une méthode n'est pas suffisant pour différencier deux méthodes portant le même nom (une erreur est alors générée à la compilation) :

```
void methode( int i ) {
        ...
}
int methode( int i ) {
        ...
}
```

Emploi de la surcharge : un utilisateur dispose de plusieurs méthodes réalisant le même traitement pour des arguments différents.

La portée des variables

```
public class Classe{

	public void methode1(){
		int i = 0;
		System.out.println( i );
	}

	public void methode2( int i ){
		i++;
		System.out.println( i );
	}

	public static void main( String [] argv ) {

		Classe classe = new Classe();
		int i=4;
		classe.methode1();
		classe.methode2( i );
		System.out.println( i );
	}

}
```

Les bibilotèques du langages : les chaînes de caractères

```
public class Classe{

	public static void main( String [] argv ) {

		String s1 = new String( "Bonjour" );
		String s2 = new String( "Bonjour" );
		if( s1.equals( s2 ) ){
			System.out.println( "égales" );
		}

		s2 = s1.substring( 3 );
		s1.concat( s2 );

		int index = s1.indexOf( "j" );

		// ...
	}

}
```

Les bibilotèques du langages : la classe Math

```
public class Classe{
	public static void main( String [] argv ) {
		double pi = Math.PI;
		double sinusDePI = Math.sin( pi/4 );
		double arrondi = Math.floor( sinusDePI );
		double arrondiAuCarré = Math.pow(arrondi, 2.0 );
		double racinneCarré = Math.sqrt( arrondiAuCarré );
		double moinsArrondiAuCarré = -arrondiAuCarré;
		double valeurAbsolue = Math.abs( moinsArrondiAuCarré );
		double enDegrés = Math.toDegrees( valeurAbsolue );
		// ...
	}
}
```

Les bibilotèques du langages : la package java.util (1/3)

```
import java.util.*;

public class Classe{

	public static void main( String [] argv ) {

		Random générateur = new Random();
		double nombre = générateur.nextDouble();

		Calendar aujourdhui = Calendar.getInstance();
		int jourDuMois = aujourdhui.get( Calendar.DAY_OF_MONTH );

		//...

	}

}
```

Les bibilotèques du langages : la package java.util (2/3)

```
import java.util.*;

public class Classe{
	public static void main( String [] argv ) {

		String s1 = new String( "chaîne 1" );
		String s2 = new String( "chaîne 2" );

		ArrayList<String> tab = new ArrayList<String>();

		 tab.add( s1 );
		 tab.add( s2 );

		String s = vecteur.get( 0 );

	}
}
```

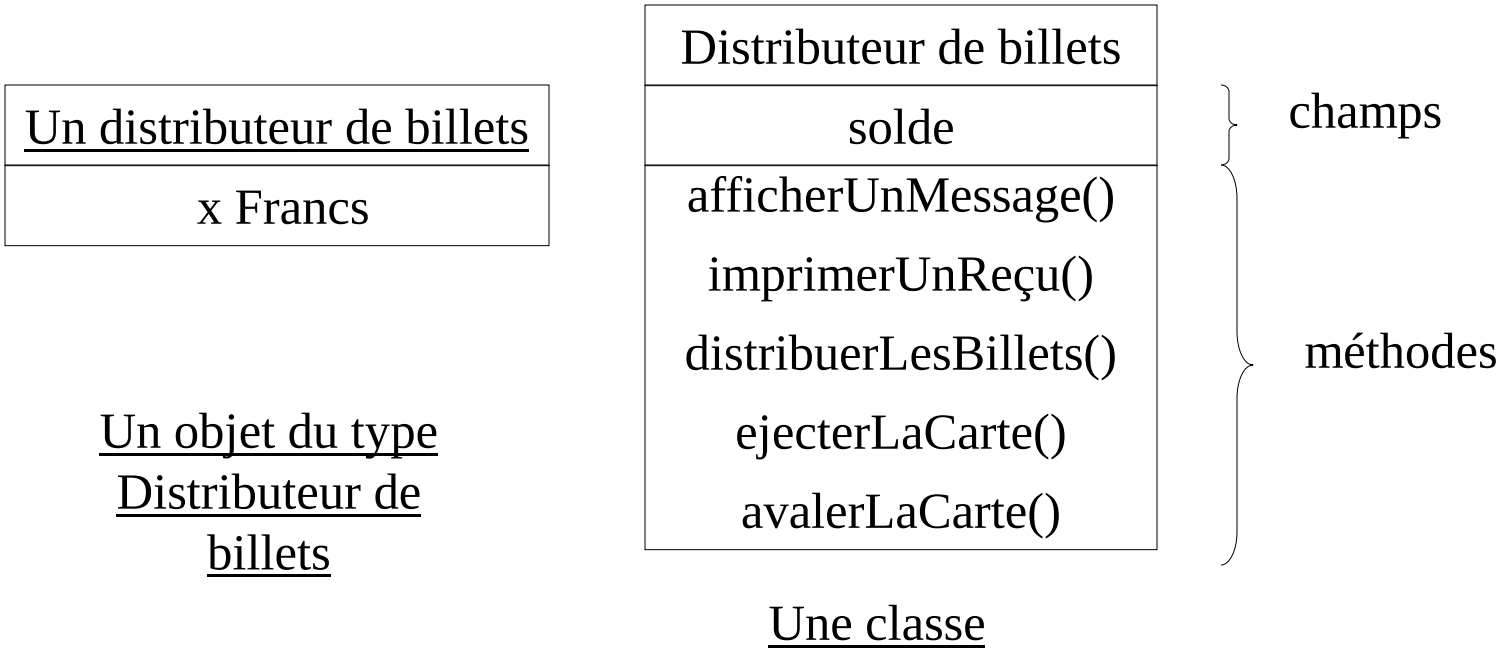
Les bibilotèques du langages : la package java.util (3/3)

```
import java.util.*;

public class Classe{
	public static void main( String [] argv ) {


		String s1 = new String( "chaîne 1" );
		Integer i = new Interger( 1 );

		Hashtable<Integer,String> tableDeHachage = new Hashtable<Interger,String>();
		 tableDeHachage.put( i, s1 );

		String s = (String) tableDeHachage.get( i );


	}
}
```

L’approche objet de Java

Notion de classe et d'objet

L'état d'un objet (champs) est caché en son sein. La seule façon de lire ou de changer l'état d'un objet est de lui envoyer un message (appeler une méthode).



Principe d'encapsulation des données : pour accéder aux champs, il faut passer par des méthodes.

Les classes et les objets

```
class Distributeur {
      int solde  = 1000 ;
}
```

Distributeur
solde

Les variables d'instances (champs d'une classe) reçoivent des valeurs par défaut si elles n'ont pas été initialisées lors de leur déclaration :

- 0 pour les types numériques ;
- \0 pour les chaînes de caractères ;
- false pour les booléens.

```
class ClientDistributeur {
      public static void main( String [] argv ) {

            Distributeur distributeur = new Distributeur() ;
      }
}
```

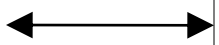
distributeur
1000

création d'un objet

Les classes et les objets

```
class Distributeur {
      int solde  = 1000 ;
}
```

une classe



Distributeur
solde

```
class ClientDistributeur {
      public static void main( String [] argv ) {

            Distributeur distributeur = new Distributeur() ; //
```

distributeur
1000

création d'un objet

```
            distributeur.solde = -10 ;
```

```
// un client ruine la banque !!
      }
}
```

Pas de restriction sur l'accès aux champs

Le mot clef private

L'idée est de ne pas confier à n'importe qui l'accès aux champs importantes. Ainsi, les données sont souvent cachées à un utilisateur.

```
class Distributeur {
      private int solde  = 1000 ;
}
```

Le mot clef « private » cache les données à un utilisateur

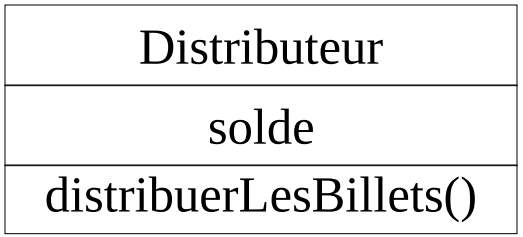
```
class ClientDistributeur {

      public static void main( String [] argv ) {
            Distributeur distributeur = new Distributeur() ;
            distributeur.solde = -10 ;
      }
}
```

// **erreur à la compilation**

Les méthodes des classes et le mot clef public

- Pour garantir l'intégrité des données tout en pouvant les modifier, il faut forcer l'utilisateur à passer par des méthodes qu'on a écrite pour lui.

```
class Distributeur {
        private int solde  = 1000 ;

        public int distribuerLesBillets( int montant ) {
                if( solde >= montant ){
                        solde = solde - montant ;
                        return montant ;
                }
                else return 0 ;
        }
}
class ClientDistributeur {

        public static void main( String [] argv ) {
                Distributeur distributeur = new Distributeur() ;
                int billets ;
                billets = distributeur. distribuerLesBillets( 100 ) ;
        }
}
```



Les packages

- Un package est un ensemble de classes ;
- Les classes d'un même package sont compilées dans un répertoire portant le nom du package ;
- Le mot clef package doit apparaître en premier dans un fichier source.

Fichier Distributeur.java

```
package banque ;

class Distributeur {
        int solde  = 1000 ;
}
```

Recommandation :
Ne mettre qu'une classe par fichier.

javac -d . Distributeur.java // compilation du fichier Distributeur.java

⇒
- Dans le répertoire courant : Distributeur.java ;
- Dans le repertoire ./banque : Distributeur.class

Le mot clef package

Par défaut, un champ est accessible depuis le code situé dans des fichiers différents du même package.

```
package banque ;

class Distributeur {
      int solde  = 1000 ;
}
```

```
package banque ;

class Banque {
      public static void main( String [] argv ) {
            Distributeur distributeur = new Distributeur() ;
            distributeur.solde = distributeur.solde + 100 ;
      }
}
```

```
package client ;
import banque ;
class ClientDistributeur {

      public static void main( String [] argv ) {
            Distributeur distributeur = new Distributeur() ;
            distributeur.solde = -100 ;    // erreur à la compilation
      }
}
```

Les champs sans modificateur de visibilité ne sont pas accessibles depuis un autre package.

Le concepteur et l'utilisateur de la classe

Le concepteur de la classe

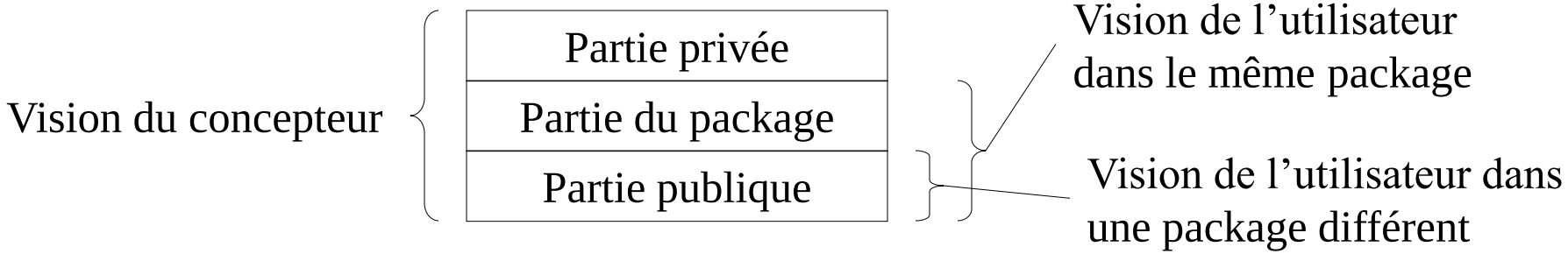
fichier : *.java

```
class Classe {
    private int champs ;
    public void methode() { … } ;

    public static void main(String[]a) {
      Classe refObjet = new Classe() ;
      refObjet.champs = 0 ;
    }
}
```

L'utilisateur de la classe

fichier : *.java

```
class ... {

    public static void main(String[] a) {
      Classe refObjet = new Classe() ;
      refObjet.methode() ;
    }
}
```



Notion de constructeur

Les constructeurs

- Un constructeur ne peut pas être déclaré abstract, synchronized ou final ;
- Un constructeur peut appeler un autre constructeur de la même classe placé avant lui en utilisant this (qui doit être la première instruction) :

```
public class Voiture {
        private int nombrePortes ;
        private String marque ;

        public Voiture( String marque, int nombrePortes) {   // 1er constructeur
                this.marque = marque ;
                this.nombrePortes = nombrePortes;
        }
        public Voiture( String marque ) {              // 2ieme constructeur
                this( marque, 5 ) ;
        }
}
```

La destruction d'objets

- La libération des ressources acquises avec un new est automatique ; la mémoire d'un objet qui n'est plus référencé peut être récupérée à tout moment :

```
class Voiture {
        // ...
        public static void main( String [] argv ) {
                Voiture voiture1 = new Voiture("Peugeot", 5) ;
                Voiture voiture2 = new Voiture("Peugeot") ;
                // …
                voiture1 = null ;           // la première voiture n'est plus référencée
        }                                   // la deuxième voiture n'est plus référencée
}
```

- On peut désactiver le ramasse-miettes avec l'option -noasyncgc de l'interpréteur Java du JDK ;
- Une application peut lancer le ramasse-miettes avec System.gc() ;

La destruction d'objets

- Si la fonction finalize est définie, elle est appelée une seule fois avant que la mémoire ne soit récupérée ;
- Les ressources non Java doivent être libérées explicitement (les fichiers par exemple) :

```
class Fichier {
        private FileInputStream fichier ;

        public Fichier( String nom ) {
                fichier = new FileInputStream( nom ) ;              // ouvre un fichier
        }

        public void close() {                                       // ferme le fichier
                if( fichier != null ) {
                        fichier.close() ;
                        fichier = null ;
                }
        }

        protected void finalize() throws Throwable {
                super.finalize() ;    // prévoir que Fichier peut devenir une sous-classe
                close() ;
        }
}
```

Membres statiques

Static : partager un membre entre toutes les instances d'une classe

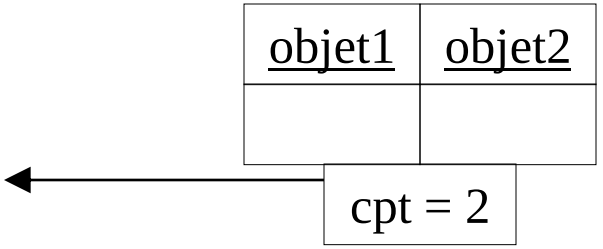
- Chaque objet à ses propres données membres :

objet1	objet2
i = 1	i = 2

```
class Compteur {
        private int i ;

        public Compteur( int i ) {
                this.i = i ;
        }
        public static void main() {
                Compteur objet1 = new Compteur( 1 ) ;
                Compteur objet2 = new Compteur( 2 ) ;
        }
}
```

- Mot clef static :

```
class Compteur {
        static private int cpt ;

        public Compteur() {
                cpt++ ;
        }
        protected void finalize() throws Throwable {
                super.finalize() ;
                cpt-- ;
        }
        public static void main() {
                Compteur objet1 = new Compteur() ;
                Compteur objet2 = new Compteur() ;
        }
}
```



Méthode statique

- commune à toutes les instances d'une classe ;
- ne peuvent accéder qu'aux données membres statiques.

```
class Compteur {
        static private int cpt ;

        public Compteur() {
                cpt++ ;
        }
        static int combien() {                          // méthode statique
                return cpt ;
        }
        protected void finalize() throws Throwable {
                super.finalize() ;
                cpt-- ;
        }
        public static void main() {
                Compteur objet1 = new Compteur() ;
                int cpt = Compteur.combien() ;          // méthode appelée par un nom
        }                                               // de classe
}
```

Exemple

Exemple d'une classe pour gérer des nombres complexes

Commencer par définir les champs :

deux flottants pour les parties réelle et imaginaire.

```
class Complexe {
        private float reel ;
        private float imag ;
}
```

Se placer du coté utilisateur pour définir des fonctions membres utiles :

```
public static void main( String [] argv ) {
        Complexe z1 = new Complexe( 1, (float)2.2 ) ;
        Complexe z2 = new Complexe( 5 ) ;
}
```

Pour pouvoir initialiser les données membres, il faut définir un constructeur

Classe nombres complexes : constructeur

```
class Complexe {
        private float reel ;
        private float imag ;

        public Complexe( float reel, float imag ) {
                this.reel = reel ;
                this.imag = imag ;
        }
        public Complexe( float reel ) {
                this( reel, 0 ) ;
        }

        public static void main( String [] argv ) {
                Complexe z1 = new Complexe( 1, (float)2.2 ) ;
                Complexe z2 = new Complexe( 5 ) ;
        }
}
```

Pour chaque objet créé il y a appel du constructeur :

z1
1
2.2

z2
5
0

Classe nombres complexes : ascesseurs

Pour respecter le principe d'encapsulation :
offrir à l'utilisateur, des accès aux données membres reel et imag :

```
class Complexe {
        private float reel ;
        private float imag ;

        // ...

        public void setReel( float x) {
                if( x < 0) reel = 0 ;
                else reel = x ;
        }
        public float getReel() {
                return reel ;
        }
        public static void main( String [] argv ) {
                CComplexe z ;
                z.setReel( 1 ) ;                // reel de z vaut 1
                float r = z.getReel() ;         // r = 1
        }
}
```

Classe nombres complexes : addition de deux vecteurs

En partant d'un programme principal :

```
... main( … ) {
        Complexe z1 = new Complexe( 1.2, 1 ) ;
        Complexe z2 = new Complexe( 3, 1 ) ;
        Complexe z3 = z1.plus( z2 ) ;
}
class Complexe
{
    private float reel ;
    private float imag ;

    public Complexe plus( Complexe z ) {
        Complexe res = new Complexe( reel+z.reel, imag+z.imag ) ;
        return res ;
    }
}
```

Passage par valeur de la référence z2

reel et imag sont les parties réelles et imaginaires de z1

Classe nombres complexes

Définition d'une fonction permettant de savoir lequel, parmi deux vecteurs, à la plus grande partie imaginaire.

```
 ... main( … ) {
        Complexe z1 = new Complexe( 1.2, 1 ) ;
        Complexe z2 = new Complexe( 3, 1 ) ;
        Complexe z3 = z1.partieImgPlusGrandeQue( z2 ) ;
 }


class Complexe {
       private float reel ;
       private float imag ;

       public Complexe partieImgPlusGrandeQue( Complexe z) {
             if( imag > z.imag ) return this ;
             return z ;
       }
}
```