1 Premier programme

```
// My first program (Hello.java)
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Compilation simple : `javac Hello.java`
Exécution : `java Hello`

2 Types de données (hors classes)

- **Types de base (primitifs) :** `int` (entier signé), `char` (caractère, sur 1 octet), `long`, `double`, …`boolean` (constantes `true, false`).
- Une classe d'emballage (*wrapper* en anglais) permet d'englober une valeur d'un type primitif dans un objet :

```
Double d1 = 34354.232;
//accès aux méthodes de Double: max, compare, ...
```

- Les *enums* comme en C :

```
public enum Humeur{TRANQUILLE, JOYEUX, TRISTE}
```

- Les constantes :

```
final int MYCONST 42;
```

Conversion de types numériques Comme en C, mais il est recommandé d'utiliser les classes *wrapper*.

3 Syntaxe de contrôle

Syntaxe C-like :

```
if (condition) {..} else {...}
while (condition) {...}
for (int i = ..; i<.. ; i++) {...}
```

Parcours "for each" à privilégier, ex :

```
ArrayList<String> fruits = new ArrayList<>();
for (String f: fruits){ //usage de f }
```

4 Exceptions

- Dérivent de la classe **Exception** : IOException, NullPointerException, ArithmeticException, …
- Levée d'exception : mot clé `throw`
- Exception pouvant être levée dans la méthode/classe :

```
public void maMethode(...) throws IOException
```

Traitement : `try-catch-(finally)`

```
try {
  FileReader lecteur = new FileReader(nomDeFichier);
  // traitement ``normal''
} catch (FileNotFoundException e) {
  // cas d'exception
} finally {
  // optionnel, traitement de tous les cas.
}
```

5 Chaînes de caractères (String)

Quelques méthodes de la classe String.

```
String s2= "abcdef"; // déclaration-initialisation
int indice = s2.indexOf('c'); //indice vaut 2.
boolean b = s2.contains("z");  // false
String s3 = s2.substring(2,4); // s3 vaut "cd"
```

Conversions :

```
String s1= "123";
int a = Integer.parseInt(s1);// l'entier 123
Integer b = Integer.valueOf(s1); // 123, emballé.
String str = String.valueOf(42); // "42"
```

6 Collections

L'interface List (éléments ordonnés)
L'interface impose l'implémentation d'un certain nombre de méthodes (on remarquera que les objets sont indexés), dont :
- `int size()`
- `Object get(int index)`.

Deux implémentations : les tableaux redimensionnables `ArrayList` ou listes chaînées `LinkedList`.

```
// une liste chaînée de Chaînes
List<String> fruits = new LinkedList<>();
fruits.addFirst("Banane") // ajout en temps constant

// Un tableau redimensionnable de Voitures
List<Voiture> voitures = new ArrayList<>();
```

L'interface Set (sans répétition) Plusieurs implémentations : `HashSet`, `TreeSet`.

```
Set<Integer> numbers = new HashSet<>();
numbers.add(...);
```

L'interface Map (clé/objet)

```
Map<String, Integer> stock = new HashMap<>();
stock.put("Pommes", 5);// 5 pommes dans mon stock
```

7 Classes, méthodes

Une classe :

```
package fr.esisar.tp1; // unité de nom
// imports éventuels (mot clé import)
public class Vehicule { // déclaration de classe Vehicule
  private int nbPlaces; // attribut
  public Vehicule(){ .. }  //constructeur
  // méthode (accesseur)
  public int getNbPlaces(){return nbPlaces;}
  public String toString() {//permet l'impression..}
}
```

Son instanciation :

```
Vehicule v = new Vehicule(); // création de l'objet v
```

Portées
- pour les classes : `public`/au niveau package sinon.
- pour les méthodes, attributs, constructeurs : `public` (tout !), `private` (dans la classe), défault (sans mot clé, au niveau package), `protected` (package et sous-classe)
- `static` : variable ou méthode de classe (partagée par toutes les instances de la classe).
- `final` : variable ou méthode immutable.

8 Héritage

Ce mécanisme permet d'hériter des attributs/méthodes d'une classe (mère) dans une autre classe (classe dérivée). Un objet dérivé possède le type de sa classe mère.

```
public class Automobile extends Vehicule{
  // accès aux méthodes de Vehicule
}

Vehicule v = new Automobile();
```

Le mot-clé `super` permet d'accéder aux membres de la super-classe.

9 Classe abstraite

C'est une classe non instanciable. Elle permet de faire une implémentation partielle.

```
// Pour "forcer" un véhicule à être une Voiture ou Velo ou ...
protected abstract class Vehicule {
  // méthodes communes à tous les véhicules
}

Vehicule v = new Vehicule(); // erreur à la compilation
```

10 Interface

Une interface représente un contrat. Toutes les méthodes sont abtraites et les attributs constants.

```
public interface AUnGuidon {
  // Les méthodes spécifiques
}

public class Velo implements AUnGuidon {
  // implémentations de ces méthodes + autres
}
AUnGuidon v = new Velo();
// méthodes appelables: uniquement celles de l'interface
```