MEMENTO JAVA

- Classes + constructeurs : **MyClass**
- Methodes,variables,instances : **myMethod**
- Constantes : **MYCONST**
- Commentaires : **// et /* */**
- Import : **import package ;**
- main : **public static void main(String[] args)**
- this : reference l'objet courant

```
switch (selecteur) :
    case 1 : instruction ; break ;
    case 2 : instruction ; break ;
    default : instruction ;
return,break,continue,if;else if ;else ;while;do while ;; for ; &  |! = (?:)  ^
```

<----------

Classes standards

- **Types** : **int**, **double**, **boolean**, **char**, byte, short, long , float
- **Math** : E,PI, random(), sqrt(n),pow(n,exp),abs(n),ceil(n),floor(n),round(n)
- **String** : length(), equals(), compareTo(), replace(old, new),split(regex), substring(beg,end), trim()(no spaces), charAt(index), indexOf(char),lastIndexOf(char),toCharArray(),copyValueof(char[]),valueOf(double), System.out.print/ln(), String.format
  Char aléatoire : (char)(Math.random()*25+97)
- **ArrayList** : get(int), remove(int), add(E)(end), size, isEmpty, contains(E), add(index,E),set(index,E),indexOf(E),clear(), toArray → E[]
- **Thread** : sleep(ms) (try-catch)
- **Scanner sc** = new Scanner(System.in) ; sc.nextType(). Java.util.scanner Enveloppe.parseType(type), convertir un string en type(String[] args)
- Double.POSITIVE_INFINITY ; (utile pour calcul minimum)
- **IntStream**, range(x,y).forEachOrdered(n->{...}) ;

<----------

Equals

- Object1 == Object2 teste l'egalité referentielle ;
- public Object **clone**{return new Object(this.args)}
- public Object(Object o){arg1=o.arg1} ou {this(o.arg1,...)}
- boolean egalité_non_referentielle(Object o) { return arg1==o .arg1 && … ;}
- boolean equals(Object o)

```
{
    if(this==o) return true ;
    if(o==null) return false :
    if(getClass()!=o.getClass()) return false ;
    ClasseFille other = (ClasseFille) o :
    if(arg1!= o.arg1) return false ;
    …
    return true ;
}
```

<----------

Objets complexes

clonage recursif : Voiture clone(){ return new Voiture(args.clone()) }
equals : Utiliser equals sur les arguments plutôt que '='

<----------->

Tableaux

- type[] nomVar = new type[taille] ;
- longueur : nomVar.length
- type[] nomVar = {…} ou en 2 lignes nomVar = new type[]{...}
- certaines methodes listées dans la classe Arrays et System(.arraycopy(taba,start,tab2,start,len)
- for(type nom : tableau)
-
- import java.util.Arraylist
- Arraylist<Class> nom = new ArrayList<>() :

type[][] nom > 2D ; nom.length pour nombre lignes ; nom[i].length pour nombre colonnes de chaque ligne

<----------->

Exceptions, Static, Final

Static :
-Ne depend pas d'un objet, appel de methodes/attribut independament des instances. Initialisation en dehors du constructeur. Si classe outil alors constructeur private.
-Instances voient static mais pas l'inverse ! GetVarInstance ne peut pas etre static.
-Class.staticMethod() → appel

Final :
-Variable non modifiables après initialisation
-Methodes ne peut pas etre redefinie dans des classes filles ; Classes ne peut pas etre étendue

Exceptions :
-public class Myexception extends Exception{ public MyException (String msg) { super(« » +msg);}
try { instructions
}catch(Exception specialisée | Exception e){ … } //Avec deux catch, d'abord les exceptions perso
finally{...}
Methode throws <TypeException>{ if(...) throw new <typeException>}
@Override, Deprecated, SupressWarnings

<----------->

Heritage

super :

- constructeur → super(args1,args2,...,argsn), si super() alors appel implicite
- methode → appel méthode super-classe

public class Fille **extends** classMere{...}. Herité methodes publiques, protegées sauf constructeur
-protected : recouvre toutes les classes heritées

-Substomption : Mere m = new Fille(...). Variable Mere et instance Fille. Les methodes redefinies sont appelés dans Fille, les methodes de Fille ne sont pas accessibles.
Lambda y = new Lambda() ; y.maMethode(Point p ou toute classe heritée de point)

-Surcharge de methodes, seul les arguments doivent etre differents. La JVM regarde le type de l'instance pour choisir la methode à appeler en cas de redefinition. La JVM ne regarde

pas les instances des arguments

-Cast : instanceof , getClass(), String getType(à definir).
Lors de cast, verifier les instances, securiser le cast avec un if(instanceof)

-Abstract : Classes abstraites non instanciables (mais la substomption fonctionnne).
Les methodes abstraites doivent etre implementés dans les classes filles et constructeur protected !

Gerer l'info de type :
Solution 1 : getNom() dans chaque classe
Solution 2 : getNom() dans classe mère et constructeur avec String. Classe fille donnent leur nom au constructeur
Solution 3 : abstract getNom()

-Interfaces : keyword **implements**, une classe peut implementer plusieurs interfaces.
Prototypes des interfaces sont publiques et doivent etre implementés dans la classe qui implemente l'interface.

<--->

Packages

public → visible partout
protected → même paquet et classes filles
privé → classe seule
∅ → même paquet

<--->

FILES

<--->

DESIGN PATTERN & GENERICITE

<--->

FORMULES

- **Math.random()<probability**
- **Math.random()%(MAX-MIN+1)+MIN → nombre entre min et max**
- **sqrt(Math.pow(xa-xb,2)+Math.pow(ya-yb,2)) → distance**
- **(i(+-1)+taille)%taille → Thoricité**
- **Variable globale(taille,nom,couleur...), deux methodes :**
  - **->interface avec variable static final int NOM;**
  - **->une variable(traditionelle, args[i] ou Scanner) dans une seule classe**