P.O.O. JAVA

Licence NEC

Mike Deguilhem (mike_deguilhem@yahoo.fr)

Plan du cours

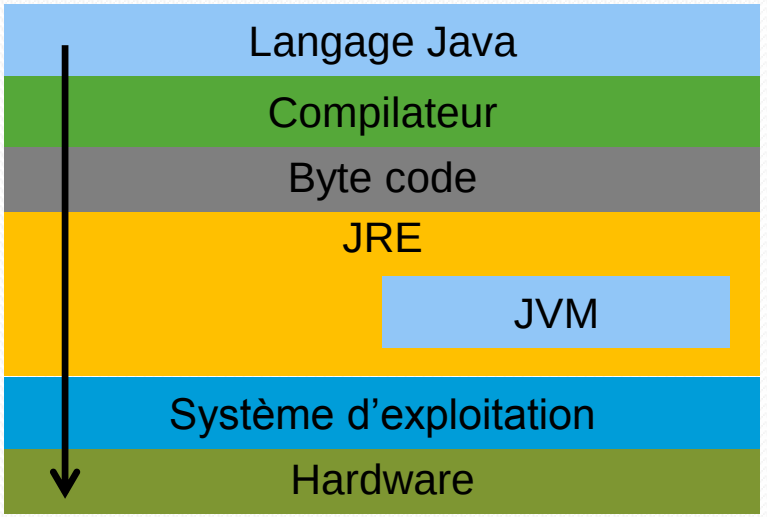
- Java
- Typage des données
- Tableaux
- Instructions de contrôle
- Rappels P.O.O.
- Objets et notions associées
- Héritage et notions associées
- Généricité
- Transcription UML -> Java
- Exceptions
- Paquetages
- Classes abstraites
- Interfaces
- JUnit

Java

- Historique
  - technique informatique développée initialement (années 1990) par Sun Microsystems puis par Oracle
  - utilisé dans une grande variété de plates-formes depuis les systèmes embarqués et les téléphones mobiles, les ordinateurs individuels, les serveurs, les applications d'entreprise, les superordinateurs
  - souhait de développer un langage de programmation indépendant de la plate-forme hardware.

Java

- Ensemble d'éléments techniques
  - Le langage Java
  - Le compilateur
  - La machine virtuelle (JVM)
  - Environnement d'exécution Java (JRE)



Prologue

- Les commentaires en Java
  Deux notations : // ou /* */
  - // la suite de la ligne est un commentaire
  - /* tout ce qui suit est un commentaire. cela implique que ce commentaire se poursuit sur plusieurs lignes.
    Là, je termine le commentaire par */
- Terminaison d'instruction : ;
- Les blocs d'instructions : {}
- Attention à la casse et aux conventions de nommage

Typage des données

- Utilisation des variables
  - déclaration
    - typeDeDonnee nomVariable; // déclaration
  - utilisation
    - nomVariable = valeur; //utilisation de ma variable si déjà déclarée
  - récupération de valeur
    - typeDeDonnee nomVariable = valeur; // déclaration et initialisation
    - typeDeDonnee autreNomVariable = nomVariable
- Constantes : mot clé « final »
  - final typeDeDonnee NOM_CONSTANTE = valeurConstante;

Typage des données

- Type primitifs
  - Booléen : boolean (true, false)
  - Caractère : char (caractère unicode, 16 bits, noté entre ‘’)
    - Exemple : lettre = ‘A’;
  - Entiers : byte(8 bits), short(16), int(32), long(64)
  - Réels : float(32), double(64)

- Le cas particulier chaine de caractères
  - Objet utilisé comme un type primitif
  - String prenom = "hector" ;

Tableaux : Notation [ ]

- Déclaration
  - double montantsJournaliers[ ];
  - Ou double[ ] montantsJournaliers;
  - Pas d'indication de taille dans la déclaration
- Instantiation
  - double montantsJournaliers[ ] = new double[31];
  - double montantsJournaliers[ ] = {250.50 , … , 400.85}
- Plusieurs dimensions
  - double montantsMensuels[ ][ ] = new double[12][31];
- Taille du tableau : méthode length

Instructions de contrôle

- Condition : mots clés : if / else / else if
  - if (condition) { …
    }
    else { … // bloc else facultatif
    }
  - if (conditionA) { …
    }
    else if (conditionB) { …
    }
    else { …
    }

Instructions de contrôle

- Choix : mot clé « switch »
  - switch (variable) {
    case valeurA : … ; break;
    case valeurB : … ; break ;
    default : … ;
    }
  - break est nécessaire sinon, tous les blocs suivant le premier cas concordant sont exécutés même si le cas n'est pas vérifié
  - Attention à la version de Java utilisée car switch ne fonctionne que pour les types entiers et caractère jusqu'à la version 7. Java 7 permet l'utilisation de switch sur des chaines de caractères.

Instructions de contrôle

- Boucles
  - Plusieurs types de boucles
    - do {
      }
      while (condition)
    - while (condition){
      }
    - for (initialisation ; condition ; incrément) {
      }
- Branchements inconditionnels
  - break : interrompt la boucle
  - continue : passe au tour de boucle suivant

Instructions de contrôle

- Conditions
  - Égalité : ==
  - Inégalité : !=
  - Inférieur : <
  - Inférieur ou égal : <=
  - Supérieur : >
  - Supérieur ou égal : >=
  - Et logique : &&
  - Ou logique : ||

Procédure / fonctions

- Déclaration
  - visibilité typeDeDonneeRetourne nomMethode(typeDeDonneeParametre1 nomParametre1, typeDeDonneeParametre2 nomParametre2,…){
    traitements;
    }
  - Mots clés
    - Procédure : void
    - Fonction : return
  - Exemples
    - public void affiche(String message){ System.out.println(message); }
    - public int somme(int nb1, int nb2){ return nb1 + nb2; }
- utilisation
  - affiche("Hello world");
  - int resultat = somme (5 , 10);

Concepts de P.O.O.

- Séparation données/traitements
- Encapsulation des données
- Objets
  - Attributs
    - types primitifs
    - objets
  - Méthodes
- Héritage

Objets

- En Java, hormis les types primitif, tout est objet.
- Utilisation de classes qui possèdent
  - Attribut(s)
    - Déclaration : visibilité typeDeDonnee nomAttribut;
  - Méthode(s)
    - Constructeur(s)
    - Déclaration : visibilité typeRetourné nomMéthode(paramètres) {…}
- Un programme est un ensemble d'objets
  - Il fait appel à des classes
  - Une des classes possède une méthode exécutable
    - public static void main (String[] args) {…}

Objets

- Encapsulation des données : visibilité des classes et attributs
  - public
  - protected
  - private
- Corps (squelette) d'une classe
  - Déclaration de la classe{
    déclaration des constantes
    déclaration des attributs
    implémentation constructeur(s)
    implémentation méthodes
    }

Objets – création

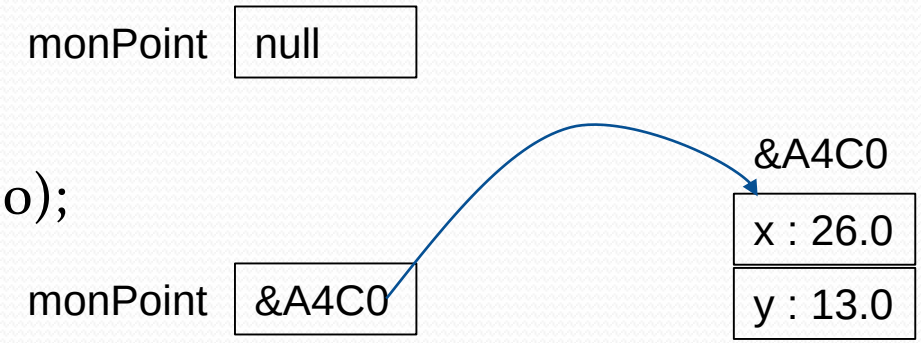
```
public class Point {
    private double x;
    private double y;

    public Point(double x, double y){ //constructeur
                this. x= x;                    // mot-clé this
                this. y= y;
    }

    //getters et setters
    public void setX(double x){ this.x = x; }
    public double getX(){ return x; }

    public void setY(double y){ this.y = y; }
    public double getY(){ return y; }
}
```

Objets - utilisation

- Instanciation : mot clé « new »
  - La déclaration d'une variable de type objet ne crée pas d'instance de cet objet.
    Point pointA;
  - La création d'une instance passe par l'appel du constructeur avec new.
    pointA = new Point(10.00 , 20.00);
- Accès aux méthodes : notation pointée
  - double abscisse = pointA.getX();
- null
  - L'abscence d'instance d'un objet est définie par le mot-clé null.
    Point pointB = null;

Objets – pointeur

- Différence entre objets et types primitifs
  - Le nom d'une variable de type objet contient un pointeur vers l'instance de l'objet en mémoire.
  - Exemples :
    - Point monPoint;
    - monPoint = new Point(26.0, 13.0);



Objets – pointeur

- Quiz :
  - Soit le code suivant

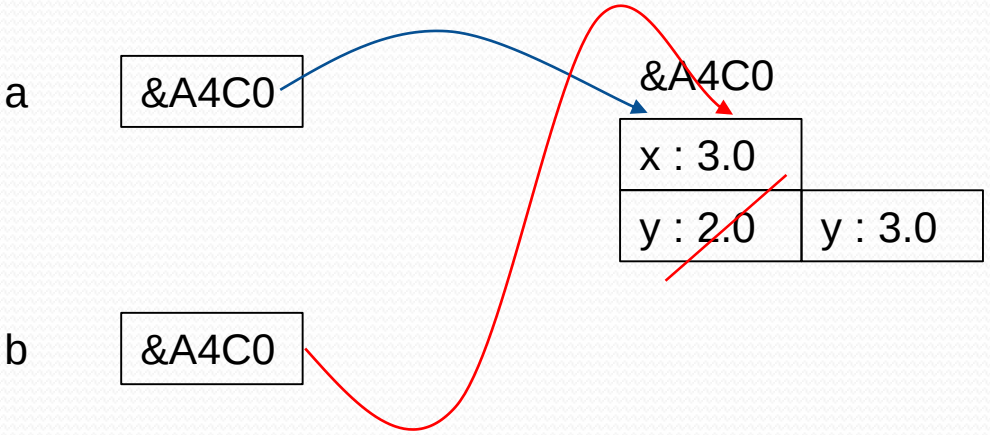
```
Point a = new Point( 3.0 , 2.0 );
Point b = a;
b.setY ( 3.0 ); // affecte 3 à l'ordonnée de b
double ordonnee = a.getY();
```

  - Quelle valeur contient la variable « ordonnee » ?
    - 1 : null
      2 : 2.0
      3 : 3.0
      4 : 0.0

Objets – pointeur

- Différence entre objets et types primitifs
  - Attention à la copie

```
Point a = new Point( 3.0 , 2.0 );
Point b = a; //b contient l'adresse contenue dans a
b.setY ( 3.0 ); // affecte 3 à l'ordonnée du seul objet
double ordonnee = a.getY(); //ordonnee contiendra 3
```



Objets - Surdéfinition

- Il est possible de définir plusieurs méthodes ayant le même nom.
  Les signatures (les paramètres, ou arguments) doivent être différentes.
- public class Individu{

```
private String nom;
private String prenom;
private String civilite;
…
public void nommer(String nom){this.nom = nom;}
public void nommer(String nom, String prenom){
        this.nom = nom;
        this.prenom = prenom;
}
public void nommer(String civilite, String nom, String prenom){
        nommer(nom, prenom);
        this. civilite = civilite;
}
```

- Cas typique d'utilisation : les constructeurs

Objets - comparaison

- Avec les types primitifs, l'opérateur de comparaison == compare les valeurs.
  Avec les objets, ce même opérateur teste si les références pointent sur la même instance d'objet. On parle alors d'opérateur d'identité.
  - Point pointA = new Point(2.0 , 3.0);
    Point pointB = new Point(2.0 , 3.0);
    if (pointA==pointB) {
    « dead code »
    }
- Pour comparer l'égalité du contenu des chaines, il existe la méthode equals (ou la méthode compareTo)
  String statut = "étudiant" ;
  if (statut.equals("étudiant")){ … }

Objets - Type statique

- Type statique
  - N'existe qu'en un seul exemplaire pour toutes les instances de la classe
  - On parle de champs et méthodes de classe
  - public class Connexion{

```
        private static int nbConnexion = 0;
        private String url;

        public static void ajouteConnexion(){
            //n'a accès qu'aux champs de classe
            nbConnexion++;
        }
}
```

  - On peut accéder à une méthode de classe sans instancier la classe
    Connexion.ajouteConnexion();

Objets – ramasse miette

- Java récupère automatiquement la mémoire allouée aux objets inutilisés.

- Point a = new Point(2.0 , 3.0); //allocation de mémoire pour un objet Point
  a = new Point (10.0 , 15.0); // a contient la référence d'un autre objet

  A l'activation du ramasse-miette, l'instance de l'objet de coordonnées (2.0 , 3.0) sera effacée de la mémoire, la quantité de mémoire qui lui été affectée est récupérée.

Héritage

- Facilite la maintenance
  - Évite la duplication du code
  - Utilisation d'un code déjà testé
- Vocabulaire
  - classe mère/fille ou super classe/sous classe
  - Spécialisation – généralisation
- Sous-classes
  - Héritent des attributs et méthodes de la superclasse
  - Ajoutent leurs propres méthodes et attributs

Héritage

- Visibilité des attributs et méthodes – Encapsulation
  - public : visible par toutes les classes
  - private : visible uniquement au sein de sa classe
  - protected : visible par les classes du package et les classes filles

- Mot-clé extends
  - public class Point3D extends Point{

```
        private double z;

        public Point3D (double x, double y, double z){
                super (x, y); //appel constructeur de la classe Point
                this.z = z;
        }

        //getters and setters, etc…
}
```

Héritage – polymorphisme

- Polymorphisme
  - Il est possible de substituer à toute instance de la superclasse, une instance de (sa) sousclasse.
    - Exemple :

```
public class Graphique{
        public void afficherPoint (Point p){
                ...
        }
}

Graphique graphe = new Graphique();
Point3D point3D = new Point3D( 3 , 2 , 5);
graphe.afficherPoint(point3D);
```

    - Moyen mnémotechnique : qui peut le plus, peut le moins !

Transtypage

- Changer le type d'un objet
- Transtypage explicite
  - double d = 5.0; float f = 7.0; long l = 6; byte b = 1;
    OK : d = f; f = l; l = b; d = (f*l) + d;
    incorrect : b = f; l = d; l = f + d;
- Par méthode
  - int j = Integer.parseInt("123");
    String s = Integer.toString(j);
- Transtypage de référence d'objet
  - Faire passer un objet pour un autre objet
    - public void affichage(Object o){
      graphe.afficherPoint((Point3D) o);
      }

Héritage – redéfinition

- Redéfinition de méthode
  - La surdéfinition implique des signatures différentes. La redéfinition reprend la même signature
  - public class Point{…
    public String toString(){ return ("x: "+x+" y: "+y);}
    }
    public class Point3D extends Point{…
    public String toString(){ return (super.toString() + " z: "+z);}
    }

- Mot-clé final
  - Une méthode déclarée avec le mot-clé final ne peut être redéfinie dans une sousclasse

Généricité

- Utiliser le même code pour différents types de valeurs
  - public class Generique <T> {
    private T maValeur;

    public Generique(T maValeur){ this.maValeur = maValeur;}

    public void setMaValeur(T maValeur){ this.maValeur = maValeur;}

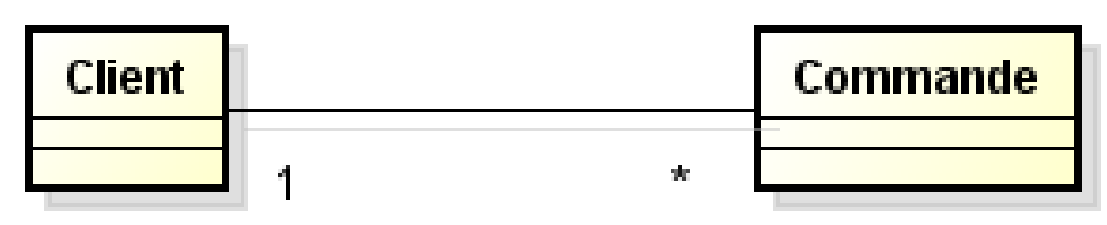
    public T getMaValeur(){ return maValeur; }
    }
  - Generique<Integer> montant = new Generique<Integer>(50);
    Generique<String> prenom = new Generique<String> ("hector");

Généricité - objets utiles

- ArrayList
  - Tableau plus aisé d'emploi / dynamique
  - List<String> adressesIP = new ArrayList<String>();
  - add / get
- HashMap
  - Couple clé/valeur
  - Map <String, Employe> employes = new HashMap<String,Employe>();
  - put / get

Transcription UML -> Java

- UML :



- Java :
  - Classe Client :
    - private HashMap<String, Commande > commandes;
    - private List<Commande> commandes;
  - Classe Commande : private Client client;

Exceptions

- Utile pour éviter les plantages dus aux erreurs
  - Traitement des exceptions
    - Bloc try-catch

```
try{ //opération qui peut générer des erreurs
        int dividende = 10;
        int diviseur = 0;
        double quotient = dividende / diviseur;
        System.out.println("Resultat:" + quotient);
}
catch (ArithmeticException e){
        System.out.println("Impossible de diviser par zéro");
}
```

Exceptions

- Clause finally
  - Un bloc try contient… un try
  - N blocs catch
  - Éventuellement un bloc finally
    - Ce bloc d'instructions est toujours exécuté, qu'une (ou plusieurs) exception ait eu lieu ou pas. Même si une exception non « catchée » s'est produite.
    - N'est pas exécuté si l'instruction System.exit() est utilisée dans le bloc try.
    - Généralement utilisé pour fermer une connexion à la base de données, fermer un fichier ouvert, etc…
- Toute méthode qui peut générer des erreurs ou problèmes est en mesure de lancer une exception qui pourra être récupérée par une méthode appelante.
  - Mot-clé throw ; mot-clé throws

Exceptions

- Les différentes exceptions forment un arbre d'héritage dont la classe mère est la classe « Exception »

- Il est possible de créer ses propres exceptions
  - extends Exception

Paquetages

- Package
  - Unicité du nom de package
- Regroupement de classes
  - Evite les doublons de noms de classes (grâce à l'unicité du nom de package).
- Mot-clé import
  - Suivi d'un nom de package pour utiliser les classes contenues dans ce package
  - import java.io.File;
  - Ou import java.io.*; //déconseillé car il vaut mieux importer les seules classes qui nous sont utiles

Classes abstraites

- Non instanciables – mot-clé abstract
- Sert de « base » pour un héritage en ne fournissant que des signatures
- Regroupe des caractéristiques communes
  - Les sousclasses ne sont instanciables que si elles définissent toutes les méthodes abstraites héritées
- Les méthodes static, final ou private ne peuvent pas être abstraites
- Peut contenir des méthodes implémentées

Classes abstraites

- public abstract class ClasseAbstraite{
  public void uneMethode(){}
  public int uneFonction(int parametre){}
  }

- ClasseAbstraite maClasse = new ClasseAbstraite(); interdit

- public class ClasseHeritee extends ClasseAbstraite{
  ....
  }

Interfaces

- Limitation de Java : pas d'héritage multiple !
- Solution : les interfaces (mot-clé interface)
  - Méthodes implicitement abstraites => déclarent des méthodes que les sousclasses doivent implémenter
  - Méthodes implicitement publiques
  - Attributs uniquement de type static et final
- Une interface est non instanciable
- Elle ne contient que des méthodes non implémentées
  - Implémentées dans les « sousclasses »
- Une sousclasse peut implémenter plusieurs interfaces

Interfaces

- public interface Affichage{
  public void affiche();
  }

- public class AffichEntier implements Affichage{
  private int valeur;
  public void affiche(){System.out.println("val:"+valeur);}
  }
- public class AffichChaine implements Affichage{
  private String valeur;
  public void affiche(){System.out.println("val:"+valeur);}
  }

JUnit

- Framework de tests unitaires
  - Automatisation et scénarisation des tests
  - Classes de tests
    - TestCase
    - TestSuite
  - Conventions de nommage :
    - nomClasseTest
    - Méthodes testNomMethodeATester
  - Tests :
    - Une méthode par test
    - Ordre d'exécution des méthodes aléatoire

JUnit

- Unité de test : assertion
  - assertEquals() / égalité de deux valeurs de type primitif ou objet
  - assertFalse() / valeur du paramètre est fausse
  - assertNull() /objet fourni en paramètre est null
  - assertNotNull() / objet fourni en paramètre n'est pas null
  - assertSame() / objets fournis font référence à la même instance
  - assertNotSame() / objets fournis ne font pas référence à la même instance
  - assertTrue()
- Méthode fail : indique que le test est raté (catch d'une exception)

JUnit

- Exemple de test
  - public class ClientTest extends TestCase {
    public void testGetRemiseNulle() {
    Client client = new Client();
    client.setAnciennete(5);
    int remise = client.getRemise(5, 500.0);
    assertEquals("Cas 01 : Remise zéro attendue", 0, remise);
    }

JUnit

- Exemple de test (suite)
  - public class TestExecute {
    public static void main(String[] args) {
    TestRunner.run(ClientTest.class);
    }
    }