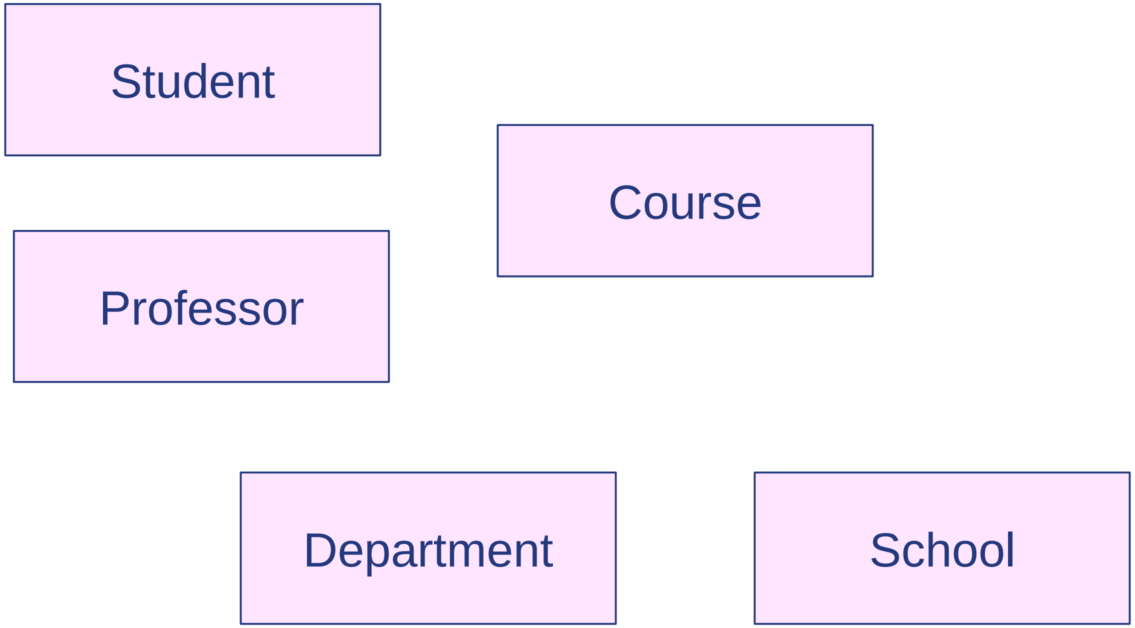
Classes - Basics

- Classes
- Information hiding
- Method overloading
- Useful methods: toString() and equals()
- Parameter passing: call by reference
- Final fields
- Static fields and methods
- Objects Initialization

Class

- A class is an unit of Java programs; that is, Java programs consist only of classes.



Student

Course

Professor

Department

School

Class

- Each class consists of fields and methods

Each class can be public or not.

```
public class Rectangle {

   private int leftTopX, leftTopY ;
   private int rightBottomX, rightBottomY ;

   public Rectangle(int x1, int y1, int x2, int y2) {
      …
   }
   public void moveBy(int deltaX, int deltaY) {
      …
   }
   public void print() {
      …
   }
   public static void main(String[] args) {
      …
   }
}
```

fields

methods

Each field and method can be public, private, or protected.

Class: Rectangle

- Methods are implemented within the class.

```
public class Rectangle {
   private int leftTopX, leftTopY ;
   private int rightBottomX, rightBottomY ;

   public Rectangle(int x1, int y1, int x2, int y2) {
      leftTopX = x1 ; leftTopY = y1 ;
      rightBottomX = x2 ; rightBottomY = y2 ;
   }
   public void moveBy(int deltaX, int deltaY) {
      leftTopX += deltaX ; rightBottomY += deltaY ;
   }
   public void print() {
      System.out.printf("(%6d,%6d), (%6d,%6d)%n",
         leftTopX, leftTopY, rightBottomX, rightBottomY) ;
   }
   public static void main(String[] args) {
      Rectangle r = new Rectangle(10, 10, 200, 400) ;
      r.print();
      r.moveBy(50, 50) ;
      r.print();
   }
}
```

Constructor is used to initialize fields

Object should be created by new operator

No-argument Constructor

- Many classes contain a constructor with no arguments that creates an object whose state is set to an appropriate default
  - Numeric values: 0, boolean: false, object variable: null
- If you write a class with no constructors whatsoever, then a no-argument constructor is provided for you.
  - This constructor sets all the instance fields to their default values.

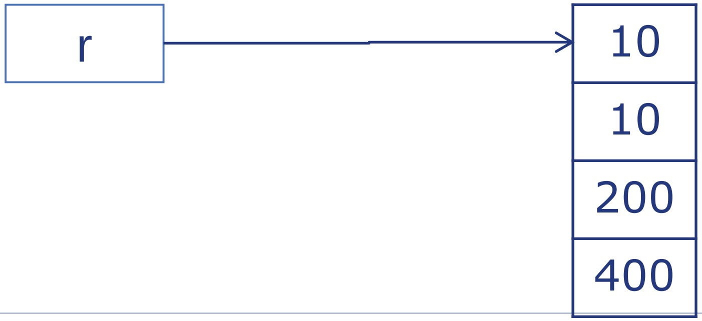
```
public class Employee {
  private int id;
  private String name;
  private double salary;
  public Employee() {
    id = 0;
    name = null;
    salary = 0.0;
  }
}
```

Object Creation

- In Java, objects can be created only through **new** operator.
  - Rectangle r(10, 10, 200, 400) is not allowed!

```
public class Rectangle {
  …
  public static void main(String[] args) {
    Rectangle r = new Rectangle(10, 10, 200, 400) ;
    r.print(); // r.method() not r->method()
  }
}
```

- Class variable points to the created object !



Class Variable

- Class variable is a reference to the created object ! It’s not an object.

```
public class Rectangle {
  private Point p ; // Error! It should be Point p = new Point()
  …
  public static void main(String[] args) {
    Rectangle r ; // Error! It should be Rectangle r = new Rectangle() ;
    r.print();
    System.out.println(p) ;
  }
}
```

- The program will
  - issue an compile-time error “The local variable r may not have been initialized” or
  - throw an exception “java.lang.NullPointerException”

Class: Summary

- Each class can be public or not.
- A class consists of fields(variables) and methods(functions).
- Each field and method can be public, private, or protected.
- All the methods should be implemented within the class.

Information Hiding

- Each field and method can be public, private, or protected.
- Only public members can be accessed from outside of the class

```
// Rectangle2.java
class Rectangle2 {
  private int leftTopX, leftTopY ;
  private int rightBottomX, rightBottomY ;
  private void setLeftTop(int x, int y) { leftTopX = x ; leftTopY = y ; }
  private void setRightBottom(int x, int y) { rightBottomX = x ; rightBottomY = y ; }

  public Rectangle2(int x1, int y1, int x2, int y2) {
    setLeftTop(x1, y1) ; setRightBottom(x2, y2) ;
  }
  public int getArea() {
    return (rightBottomX - leftTopX) * (rightBottomY - leftTopX) ;
  }
}
```

Information Hiding

```
// RectangleTest.java
class Rectangle2 { // not public class. Each source file can contain only one public class!
    private int leftTopX, leftTopY ;
    private int rightBottomX, rightBottomY ;
    private void setLeftTop(int x, int y) { leftTopX = x ; leftTopY = y ; }
    private void setRightBottom(int x, int y) { rightBottomX = x ; rightBottomY = y ; }

    public Rectangle2(int x1, int y1, int x2, int y2) { setLeftTop(x1, y1) ; setRightBottom(x2, y2) ; }
    public int getArea() { return (rightBottomX - leftTopX) * (rightBottomY - leftTopX) ; }
}
public class RectangleTest {
    public static void main(String[] args) {
        Rectangle2 r1 = new Rectangle2(0, 0, 50, 50) ;
        Rectangle2 r2 = new Rectangle2(0, 0, 100, 100) ;

        System.out.println(r1.getArea()) ;
        System.out.println(r2.getArea()) ;

        r1.setLeftTop(10, 10) ; // The method setLeftTop(int, int) from the type Rectangle2 is not visible
    }
}
```

Information Hiding

- Package is the default visibility. Package visibility will be discussed later.

```
class Rectangle2 {
    private int leftTopX, leftTopY ;
    private int rightBottomX, rightBottomY ;
    void setLeftTop(int x, int y) { leftTopX = x ; leftTopY = y ; }
    void setRightBottom(int x, int y) { rightBottomX = x ; rightBottomY = y ; }

    public Rectangle2(int x1, int y1, int x2, int y2) {
       setLeftTop(x1, y1) ; setRightBottom(x2, y2) ;
    }
    public int getArea() { return (rightBottomX - leftTopX) * (rightBottomY - leftTopX) ; }
}
public class RectangleTest {
    public static void main(String[] args) {
       Rectangle2 r1 = new Rectangle2(0, 0, 50, 50) ;
       r1.setLeftTop(10, 10) ; // OK
    }
}
```

- Package visibility is very dangerous! **Be sure to specify “private” or “public”.** Don’t leave it blank.

Overloading

- Two or more methods (including constructors) with the same name can be allowed when they have different signatures; that is, different parameter types.

```
class Rectangle3 {
   private int leftTopX, leftTopY ;
   private int rightBottomX, rightBottomY ;

   public Rectangle3(int x1, int y1, int x2, int y2) {
      leftTopX = x1 ; leftTopY = y1 ;
      rightBottomX = x2 ; rightBottomY = y2 ;
   }
   public void moveBy(int deltaX, int deltaY) {
      leftTopX += deltaX ; leftTopY += deltaY ; rightBottomX += deltaX ; rightBottomY += deltaY ;
   }
   public void moveBy(int delta) { moveBy(delta, delta) ; }
   public void print() {
      System.out.printf("(%6d,%6d), (%6d,%6d)%n", leftTopX, leftTopY, rightBottomX, rightBottomY) ;
   }
}
```

```
Rectangle3 r1 = new Rectangle3(0, 0, 50, 50) ;
r1.print() ;

r1.moveBy(10, 20) ; r1.print() ;

r1.moveBy(10) ; r1.print() ;
```

Important methods: toString() and equals()

```
class Rectangle4 {
    private int leftTopX, leftTopY ;
    private int rightBottomX, rightBottomY ;

    public Rectangle4(int x1, int y1, int x2, int y2) {
        leftTopX = x1 ; leftTopY = y1 ; rightBottomX = x2 ; rightBottomY = y2 ;
    }
    public boolean equals(Object otherRectangle) {
        if ( ! ( otherRectangle instanceof Rectangle4 ) return false ;
        Rectangle4 r = (Rectangle4) otherRectangle ; // casting from Object to Rectangle4
        return this.leftTopX == r.leftTopX && this.leftTopY == r.leftTopY &&
            this.rightBottomX == r.rightBottomX && this.rightBottomY == r.rightBottomY ;
    }
    public String toString() {
        String str =  String.format("(%6d,%6d), (%6d,%6d)",
            leftTopX, leftTopY, rightBottomX, rightBottomY) ;
        return str ;
    }
}
```

Important methods: toString() and equals()

```
public class UsefulMethods {

    public static void main(String[] args) {

       Rectangle4 r1 = new Rectangle4(0, 0, 10, 10) ;
       Rectangle4 r2 = new Rectangle4(0, 0, 10, 20) ;

       System.out.println("R1: " + r1) ;
       System.out.println("R2: " + r2) ;

       String msg =  r1.equals(r2) ? "They are the same." : "They are not the same." ;
       System.out.println(msg) ;
    }
}
```

Every object is converted into a String whenever necessary !

```
R1: (     0,      0), (    10,     10)
R2: (     0,      0), (    10,     20)
They are not the same.
```

equals(): Example

```
public class Employee {
   private String name;
   private double salary;
   private LocalDate hireDay;

   …
}
```

```
public boolean equals(Object otherObject) {
   if (otherObject == null) return false;
   if (getClass() != otherObject.getClass()) return false;
   Employee other = (Employee) otherObject;
   return name.equals(other.name)
      && salary == other.salary
      && hireDay.equals(other.hireDay);
}
```

```
public boolean equals(Object otherObject) {
   if (otherObject == null) return false;
   if (getClass() != otherObject.getClass()) return false;
   Employee other = (Employee) otherObject;
   return Objects.equal(name, other.name)
      && salary == other.salary
      && Objects.equal(hireDay, other.hireDay);
}
```

Important methods: hashcode()

- A hash code is an integer that is derived from an object.
- if x and y are two distinct objects, there should be a high probability that x.hashCode() and y.hashCode() are different
- That hash code of default implementation of hashcode() in Object class is derived from the object’s memory address.

```
public class StringHashCode {
  public static void main(String[] args) {
    String string1 = "Hello";
    System.out.println(getHash(string1) + ":" + getHash(string2));       // 69609650:99162322
    System.out.println(string1.hashCode() + ":" + string2.hashCode());  // 69609650:99162322

  }
  private static int getHash(String string) {
    int hash = 0;
    for (int i = 0; i < string.length(); i++)
      hash = 31 * hash + string.charAt(i);
    return hash;
  }
}
```

Important methods: hashcode()

- You must override hashCode() in every class that overrides equals()
- Your definitions of equals and hashCode must be compatible
  - If x.equals(y) is true, then x.hashCode() must return the same value as y.hashCode().
  - If you define Employee.equals to compare employee IDs, then the hashCode method needs to hash the IDs, not employee names or memory addresses.

hashCode()

```
public class Employee {
   private String name;
   private double salary;
   private LocalDate hireDay;
   public Employee(String name, double salary, LocalDate hireDay) {
      this.name = name;
      this.salary = salary;
      this.hireDay = hireDay;
   }
   public int hashCode1() {     // version 1
      return 7 * name.hashCode()
         + 11 * Double.valueOf(salary).hashCode()
         + 13 * hireDay.hashCode();
   }
   public int hashCode2() {     // version 2
      return 7 * Objects.hashCode(name)
         + 11 * Double.hashCode(salary)
         + 13 * Objects.hashCode(hireDay);
   }
   public int hashCode3() {     // version 3
      return Objects.hash(name, salary, hireDay);
   }
   ...
}
```

hashCode()

```
    public static void main(String[] args) {
      Employee e1  =  new Employee("Kim", 200, LocalDate.of(2019, 9, 15));
      Employee e2  =  new Employee("Kim", 201, LocalDate.of(2019, 9, 15));
      Employee e3  =  new Employee("kim", 200, LocalDate.of(2019, 9, 15));

      System.out.println(e1.hashCode1() + ":" + e2.hashCode1() + ":" + e3.hashCode1());
      System.out.println(e1.hashCode2() + ":" + e2.hashCode2()  + ":" + e3.hashCode2());
      System.out.println(e1.hashCode3() + ":" + e2.hashCode3()  + ":" + e3.hashCode3());
    }
}
```

```
-943758132:-943668020:-943542868
-943758132:-943668020:-943542868
-783759971:-783506019:-754207299
```

Parameter Passing

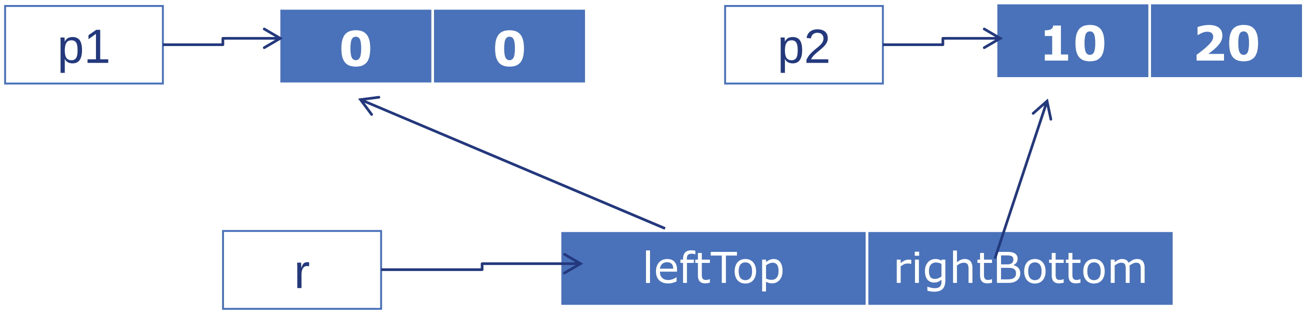
- Call by value
  - For a parameter of primitive type (int, float, …), its value is just copied to the callee.
  - Any change to a formal parameter in the callee has no impact on the formal parameter in the caller.

- Call by reference
  - Each parameter of class variable is passed by reference.
  - The reference, not the object itself is copied to the callee.
  - So, caller and callee share the same memory for the class variable

```
class Point {
    private int x, y ;
    public Point(int x, int y) { set(x, y) ; }
    public void set(int x, int y) { this.x = x ; this.y = y ; }
    public String toString() { return String.format("(%d, %d)", x, y) ; }
    public boolean equals(Object otherPoint) {
        Point p = (Point) otherPoint ;
        return x == p.x && y == p.y ;
    }
}

class Rectangle5 {
    private Point leftTop ;
    private Point rightBottom ;

    public Rectangle5(Point p1, Point p2) { leftTop = p1 ; rightBottom = p2 ; }
    public boolean equals(Object otherRectangle) {
        Rectangle5 r = (Rectangle5) otherRectangle ;
        return leftTop.equals(r.leftTop) && rightBottom.equals(r.rightBottom) ;
    }
    public String toString() { return leftTop + "," + rightBottom ; }
}
```

Each parameter of class variable is passed by reference.
Thus, leftTop and p1 refer to the same Point !

```
public class ParameterPassing {
    public static void main(String[] args) {
       Point p1 = new Point(0, 0) ;
       Point p2 = new Point(10, 20) ;

       Rectangle5 r = new Rectangle5(p1, p2) ;
       System.out.println(r) ; // (0, 0),(10, 20)

       p2.set(100, 200) ;
       System.out.println(r) ; // (0, 0),(100, 200), not (0, 0),(10, 20)
    }
}
```



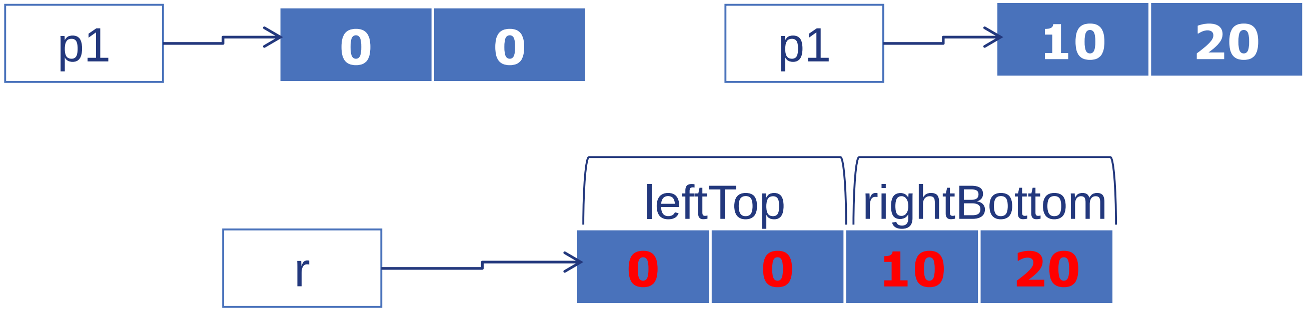
In the constructor of class Rectangle, the references are only copied!

```
     public Rectangle5(Point p1, Point p2) {
        leftTop = p1 ; rightBottom = p2 ;
     }
```

Deep Copy

- We need to copy the object itself, not the reference!

- What we want is as follows !



- Let's change the constructor of class Rectangle like this !

```
public Rectangle5(Point p1, Point p2) {
  // leftTop = p1 ; rightBottom = p2 ;
  leftTop = new Point(p1.getX(), p1.getY()) ;
  rightBottom = new Point(p2.getX(), p2.getY()) ;
}
```

```
class Rectangle6 {
    private Point leftTop ;
    private Point rightBottom ;

    public Rectangle6(Point p1, Point p2) {
        leftTop = new Point(p1.getX(), p1.getY()) ;
        rightBottom = new Point(p2.getX(), p2.getY()) ;
    }
    public boolean equals(Object otherRectangle) {
        Rectangle6 r = (Rectangle6) otherRectangle ;
        return leftTop.equals(r.leftTop) && rightBottom.equals(r.rightBottom) ;
    }
    public String toString() { return leftTop + "," + rightBottom ; }
}

public class DeepCopy {
    public static void main(String[] args) {
        Point p1 = new Point(0, 0) ;
        Point p2 = new Point(10, 20) ;

        Rectangle6 r = new Rectangle6(p1, p2) ;
        System.out.println(r) ; // (0, 0),(10, 20)

        p2.set(100, 200) ;
        System.out.println(r) ; // (0, 0),(10, 20), not (0, 0),(100, 200)
    }
}
```

Final Fields

- final fields cannot be changed after they were initialized in constructors.

```
public class Student {
    private final String name ; // name is declared as final
    private int year = 1 ;
    private String major ;

    public Student(String name, String major) {
        this.name = name ; // name can be initialized in constructor
        this.major = major ;
    }
    void setYear(int year) { this.year = year ; }
    void setName(String name) { this.name = name ; } // Not Allowed !
    void setMajor(String major) { this.major = major ; }
    public static void main(String[] args) {
        Student s1 = new Student(“James”, “Computer”) ;
        s1.setYear(2) ;
        s1.setMajor(“Mechanical”) ;
        s1.setName(“Brown”) ; // Impossible !
    }
}
```

Static Fields

- Static fields are shared by all the objects of a class.

```
class Rectangle7 {
    private Point leftTop, rightBottom ;
    public static int AllCount = 0 ;

    public Rectangle7(Point p1, Point p2) {
        AllCount ++ ;
        leftTop = new Point(p1.getX(), p1.getY()) ;
        rightBottom = new Point(p2.getX(), p2.getY()) ;
    }
    public Rectangle7() { AllCount ++ ; }
    public String toString() { return leftTop + "," + rightBottom ; }
}

public class StaticField {
    public static void main(String[] args) {
        Rectangle7 r1 = new Rectangle7() ;
        Rectangle7 r2 = new Rectangle7(new Point(0, 0), new Point(10, 20)) ;

        System.out.println(Rectangle7.AllCount) ;
        System.out.println(r1) ; System.out.println(r2) ;
    }
}
```

Constructors are also overloaded!

```
2
null,null
(0, 0),(10, 20)
```

Constant

- Public Static final is a common way to defining constants.

```
class Rectangle {
            public static final int NO_OF_SIDE = 4 ;
            ...
}
```

- More examples

java.lang.Math	
public static final double E	2.718281828459045d
public static final double PI	3.141592653589793d

java.lang.Integer	
public static final double MAX_VALUE	2147483647
public static final double MIN_VALUE	-2147483648

Static Methods

- Static methods can only access static fields.

```
class Rectangle8 {
    private Point leftTop, rightBottom ;
    private static int AllCount = 0 ;
    public static boolean noRectangle() { return AllCount == 0 ; }
    public static int getAllCount() { return AllCount ; }

    public Rectangle8(Point p1, Point p2) {
        AllCount ++ ;
        leftTop = new Point(p1.getX(), p1.getY()) ;
        rightBottom = new Point(p2.getX(), p2.getY()) ;
    }
    public Rectangle8() { AllCount ++ ; }
}
public class StaticMethod {
    public static void main(String[] args) {
        Rectangle8 r1 = new Rectangle8() ;
        Rectangle8 r2 = new Rectangle8(new Point(0, 0), new Point(10, 20)) ;

        System.out.println(Rectangle8.getAllCount()) ;
    }
}
```

Static Methods

- Standard mathematical methods in class Math are defined as public static methods.

```
class Math {
    public static double pow(double base, double exponent) { … }
    public static double abs(double argument) { … }
    public static double abs(float argument) { … }
    public static double abs(long argument) { … }
    public static double abs(int argument) { … }

    public static double min(double n1, double n2) { … }
    …

}
```

```
    if ( Math.abs(-10) == 10 ) …

    Math.min(10.5, 20) ;
```

Initialization of Objects

For the first object

1. **Static initialization block**

For each object

2. Data fields ➔ **default value**(0, false, or null)
3. **Field initializer** and **initialization block** in the order of declaration
4. **Constructor Body**

```
class Employee {
    // constructors
    public Employee(String n, double s) {/*4.*/ name = n; salary = s; }
    public Employee(double s) { this("Employee #" + nextId, s); }
    public Employee() {
        // name  = "“, salary  =1000, id initialized in initialization block
    }
    public String getName() { return name; }
    public int getId() { return id; }
    public double getSalary() { return salary ; }

    private static int nextId;
    private int id; // = 0; // 2. default value
    private String name = ""; // 3.1 instance field initialization
    private double salary = 1000 ; // 3.2 instance field initialization
    // 1. static initialization block
    static {
        Random generator = new Random();
        nextId = generator.nextInt(10000);
    }
    // 3.3 object initialization block
    { id = nextId; nextId++; }
}
```

Initialization of Objects

```java
public class Initialization {

    public static void main(String[] args) {

        Employee[] staff = new Employee[3];

        staff[0] = new Employee("Robert", 40000);
        staff[1] = new Employee(60000);
        staff[2] = new Employee();

        for (Employee e : staff)
            System.out.printf("name=%-15s,id=%6d,salary=%-10.1f%n",
                e.getName(), e.getId(), e.getSalary() );
    }
}
```

```
name=Robert         ,id=  6072,salary=40000.0
name=Employee #6073 ,id=  6073,salary=60000.0
name=               ,id=  6074,salary=10000.0
```