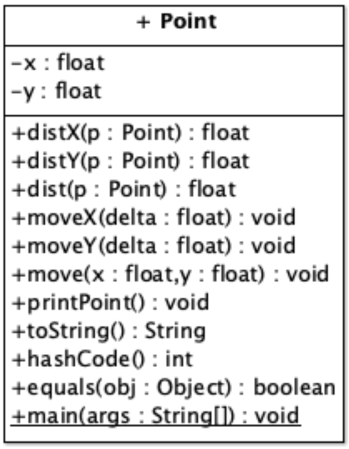
Java Classes & Primitive Types

Rui S. Moreira

@ FTC/UFP

Classes



+ Point

-x : float
-y : float

+distX(p : Point) : float
+distY(p : Point) : float
+dist(p : Point) : float
+moveX(delta : float) : void
+moveY(delta : float) : void
+move(x : float,y : float) : void
+printPoint() : void
+toString() : String
+hashCode() : int
+equals(obj : Object) : boolean
+main(args : String[]) : void

- Attributes
  - Characteristics/properties of classes
  - Primitive types (e.g., char, byte, int, float, etc.)
  - Class types (e.g., String, Date, ArrayList, Point,
  - Automatic initialisation
    - primitive type attributes are initialised to `0 or 0.0` (depending on type)
    - class type attributes are initialised to `null`
- Methods
  - Constructors (special methods with same name of class and no return value)
  - Behaviour of classes (methods receiving a list of parameters and returning a given typed value)

Example: Ponto class

```
/** Ponto Class Declaration: represents a generic point in a 2D space */
public class Ponto {
   /** Attributes: by default these attribute fields are public */
   float x = 0.0f;
   float y = 0.0f;

   /** Default Constructor: automatically provided, given there are no
   other contructors */
   //public Ponto() { }

   /* NB 1: A constructor is a public method with the same name as the
   class and with no return type;
   NB 2: if no other constructor is provided we do not need to implement
   the default constructor (that is why it is commented, above!!) */

   public static void main(String args[]){
     /* Together, the new operator followed by constructor, are used to
        create and init object attributes */
     Ponto p = new Ponto();
   }
}
```

Example: Ponto class

```
/** Ponto Class Declaration: represents generic point in 2D space */
public class Ponto {
   /** Attributes: by default these attribute fields are public */
   float x = 0.0f;
   float y = 0.0f;

   /** We may specify other specific constructors, e.g., to explicitly
   initialize attributes of our objects, e.g.,
        Ponto p = new Ponto(2.0f, 4.0f); */
   public Ponto(float a, float b) {
        x = a;
        y = b;
   }
   /** If previous constructor is provided, Java no longer provides the
   default constructor; therefore, if we want to be able to use the default
   constructor we also need to implement it as follows. */
   public Ponto() {
        x = 1.0f;
        y = 1.0f;
   }

}
```

Example: Ponto class

```
/** Ponto Class Declaration: represents generic point in 2D space */
public class Ponto {
    /** Attributes: by default these attribute fields are public */
    float x = 0.0f;
    float y = 0.0f;

    /** Specific constructors: explicitly initialize object attributes */
    public Ponto(float a, float b) {
          x = a;
          y = b;
    }
    /** Default Constructor: explicit implementation because other
    constructors exist. */
    public Ponto() {
          x = 1.0f;
          y = 1.0f;
    }

}
```

Example: Ponto class

```
/** Ponto Class Declaration: represents generic point in 2D space */
public class Ponto {
    /** Encapsulation: hide implementation with private attributes; provide
    access to attributes through well defined public method interfaces */
    private float x = 0.0f;
    private float y = 0.0f;

    /** Get attribute value */
    public float getX() {
          return this.x;
    }

    /** Set attribute value */
    public void setX(float a) {
          this.x = a;
    }

    // Please, do the same for the y attribute...
}
```

Example: Ponto class

```
/** Ponto Class Declaration: represents generic point in space */
public class Ponto {
   /** Encapsulated attributes behind public interface methods */
   private float x = 0.0f;
   private float y = 0.0f;

   /** Constructors */
   public Ponto(float a, float b) { x = a; y = b; }
   public Ponto() { x = 1.0f; y = 1.0f; }

   /** Get attribute values */
   public float getX() { return this.x; }
   public float getY() { return this.y; }

   /** Set attribute values */
   public void setX(float a) { x = a; }
   public void setY(float b) { y = b; }

   /** Other behaviour .../
   public void drawPonto(Graphics g) { /* ... */ }
}
```

Identifiers & Name Conventions

- Identifiers (for classes, interfaces, methods and attributes)
  - Are case-sensitive;
  - No size limit (number of characters);
  - Must start with: letter, underscore or $;
  - May contain digits/numbers only after first character;
- Classes & Interfaces
  - Usually first letter of each word is uppercase (Camel Case), e.g., `OneClasse`, `OneInterface`
- Methods & Attributes
  - Usually first word is entirely smallcase then first letter of remainder words is uppercase, e.g., `oneMethod()`, `oneAttribute`

Attributes: primitve types

- Logical
  - boolean (*true* or *false*);
- Textual
  - char (unsigned 16 bit number - unicode character);
  - String (final Class - sequence of unicode chars);
- Integer
  - byte (8 bits);   short (16 bits);
  - int (32 bits);   long (64 bits);
- Floating Point
  - float (32 bits floating-point)
  - double (64 bits floating-point)

Attributes: primitve types particularities

- Casts not allowed between booleans <-> integers;
- String is a class but presents also behaviour of primitive types:
  - String s = “Hello World!”; // e.g., assigned a literal sequence of chars
- Integer literals are signed (e.g., byte b; // -128 < b < 127);
- Integers are ‘int’ by default except when followed by L (e.g. long l=18L;);
- Integers may be represented in the following formats:
  - int d = 10; // decimal format
  - int o = 077; // octal format
  - int h = 0xA; // hexadecimal format
- Floating point literals may use `.` or `E`:
  - float f1 = 3.14f; // float number followed by letter `f` or `F`
  - float f2 = 6.04E23; // float number representing $6.04x10^{23}$
  - double d = 23.6E+235D; // double number followed by `d` or `D`

Some declarations

- `boolean truth = false;`
- `char a = ‘a’, c = ‘C’;`
- `char r=‘\n’, t=‘\t’; // Usage of escape char`
- `char u = ‘\u0000’; // Represents` *`null`* `char`
- `String str = new String(“string example”);`
- `String str = “string example”;//As literal`
- `byte b = 128;`
- `short s = 512;`
- `int i = 1024;`
- `long l = 2L;`
- `float f = 2.78F;`
- `double d = 8.16;//Or 8.16D;`

Memory allocation

- When declaring attributes of primitive types the memory space is automatically allocated;
- When declaring attributes of class types (e.g., String) the memory is allocated for the reference (variable) and not for the object;
- Before we can use a reference variable we must set it to point to a given object!!! (otherwise we may have a `NullPointerException`).

Creating Objects

Calling constructor:

- **1st** allocates memory for the specific object;
- **2nd** initialises attributes by default (month=0, year=0) and sets explicit attribute values (e.g. day=1);
- **3rd** runs constructor code (inside constructor method)

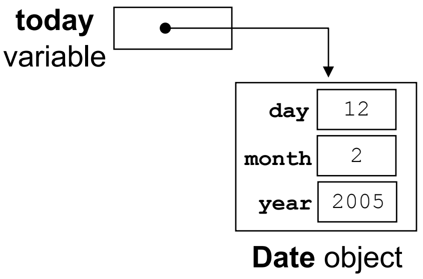
```
puclic class Date {
   int day=1, month, year;

   public Date(int d, int m, int y){
        day=d; month=m; year=y;
   }


   /** Testing class Date */
   public static void main(String args[]){
        // Declare reference variable to future object
        Date today; //today is null
        // Create object
        today = new Date(12, 2, 2005);
   }
}
```

Creating Objects

```
// Declare reference variable - today
Date today;
```



```
// today references new Date object
today = new Date(12, 2, 2005);
```

Default Initialisation of Attributes

- Java automatically initialises attributes (class member variables):
  - Primitive types:
    - `boolean` attribute variables to `false`
    - `char` attribute variables to `‘\u0000’`
    - `int` attribute variables to `0`
    - `float` attribute variables to `0.0f or 0.0F`
    - `double` attribute variables to `0.0d or 0.0D`
  - Class types:
    - *reference* attribute variables to `null`

    String s; // s is set to `null`, meaning it points to no object

Local Variables

- Default initialisation is done only for attribute variables (class member variables), not local vars;

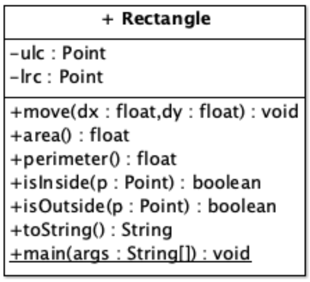
- Local variables (and parameters) are stored in the STACK which means these are NOT automatically initialised, i.e., we may NOT use local variables without explicit initialisation;

- The Java compiler generates a compile-time error if we try to use a non-initialised local variable.

Exercises

- Implement Classes:

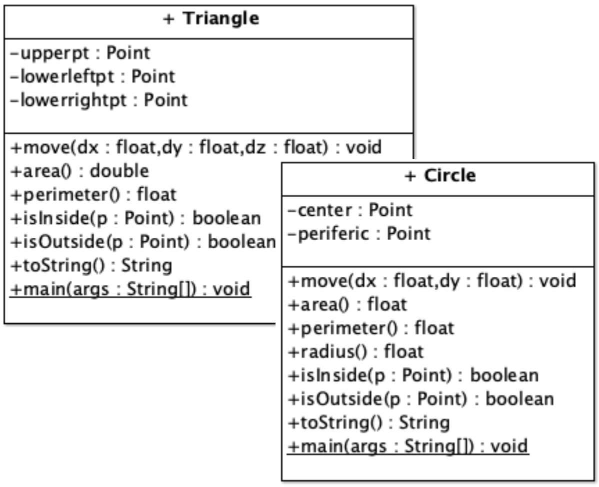
```
Rectangulo
Circulo
Triangulo
```



+ Rectangle

-ulc : Point
-lrc : Point

+move(dx : float,dy : float) : void
+area() : float
+perimeter() : float
+isInside(p : Point) : boolean
+isOutside(p : Point) : boolean
+toString() : String
+main(args : String[]) : void



+ Triangle

-upperpt : Point
-lowerleftpt : Point
-lowerrightpt : Point

+move(dx : float,dy : float,dz : float) : void
+area() : double
+perimeter() : float
+isInside(p : Point) : boolean
+isOutside(p : Point) : boolean
+toString() : String
+main(args : String[]) : void

+ Circle

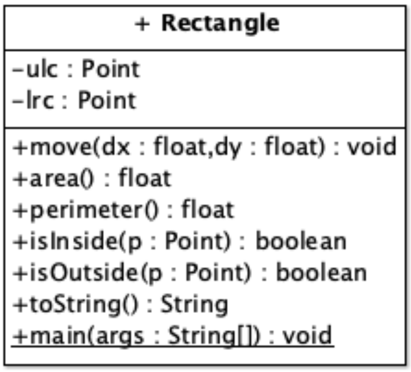
-center : Point
-periferic : Point

+move(dx : float,dy : float) : void
+area() : float
+perimeter() : float
+radius() : float
+isInside(p : Point) : boolean
+isOutside(p : Point) : boolean
+toString() : String
+main(args : String[]) : void

- Methods:

```
public void move(int dx, int dy);//Change coordinates by given amount
public boolean isInside(Ponto p);//Checks if p is inside figure
public boolean isOutside(Ponto p);//Checks if p is outside figure
public float area();//Returns area of figure
public float perimeter();//Returns perimeter of figure
public String toString();//returns string describing figure, e.g.,
  for Rectangulo with points (1,2) and (3,7) - "Rectangle@{(1,2);(3,7)}"
```

Instantiation example



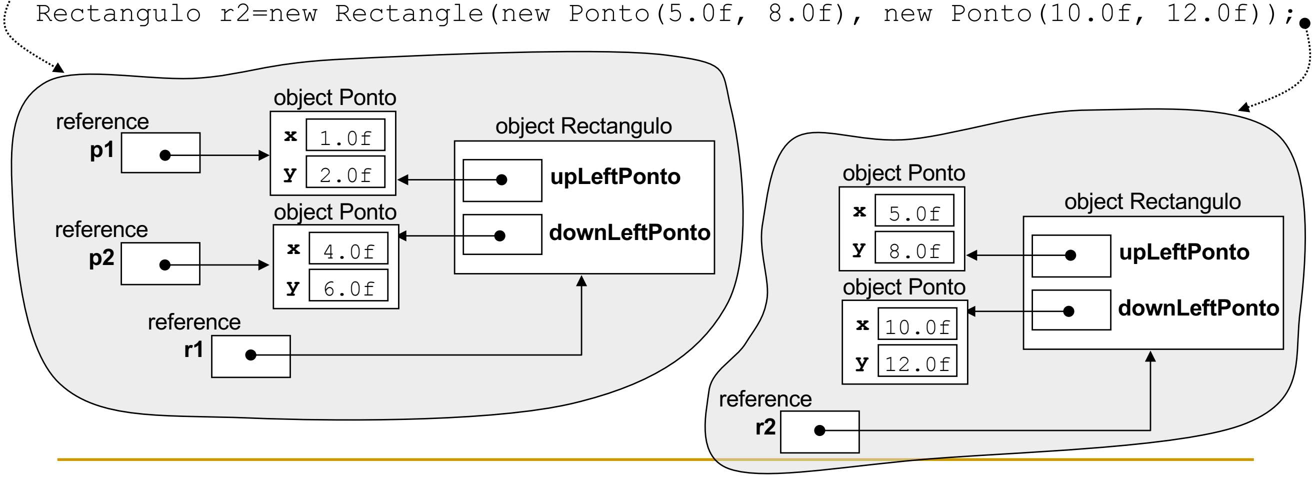
+ Rectangle

-ulc : Point
-lrc : Point

+move(dx : float,dy : float) : void
+area() : float
+perimeter() : float
+isInside(p : Point) : boolean
+isOutside(p : Point) : boolean
+toString() : String
+main(args : String[]) : void

```
// Declaring and creating 2 points
Ponto p1 = new Ponto(1.0f, 2.0f);
Ponto p2 = new Ponto(4.0f, 6.0f);

// Declaring and creating 2 rectangles
Rectangulo r1=new Rectangle(p1, p2);
Rectangulo r2=new Rectangle(new Ponto(5.0f, 8.0f), new Ponto(10.0f, 12.0f));
```



reference
p1
object Ponto
x 1.0f
y 2.0f
object Rectangulo
upLeftPonto
downLeftPonto
reference
p2
object Ponto
x 4.0f
y 6.0f
reference
r1
object Ponto
x 5.0f
y 8.0f
object Rectangulo
upLeftPonto
downLeftPonto
object Ponto
x 10.0f
y 12.0f
reference
r2