ENSF 519

2 – Introduction to Java Classes

Java Classes

- Similar to C++ with minor differences:
- Example:

```
class Car
{
        .
        .
        .
}
```

Data member and functions

The main method

- When you run a class with `java` command the class is loaded and execution starts at its main() method.
  - Other classes will be loaded, if necessary. However, their main() methods will not be used.
- Each class in an application can have its own main().
  - This is useful to use the main function as test driver.

The main Method – Command Line Argument

```
public class Test {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.print(args[i] + " ");
        System.out.println();
    }
}
```

Now if we run the class, using the following command line arguments:

```
Command line> java Test this is a test
```

The output will be: `this is a test`

Data Members and Methods

- Data members (*fields)* and member functions (*methods)* are defined inside the class declaration.
  - *Instance variables and methods* are associated with each instance of the class.
  - Methods (functions) definitions are similar to C++.

Data Members and Methods (continued)

- Example:

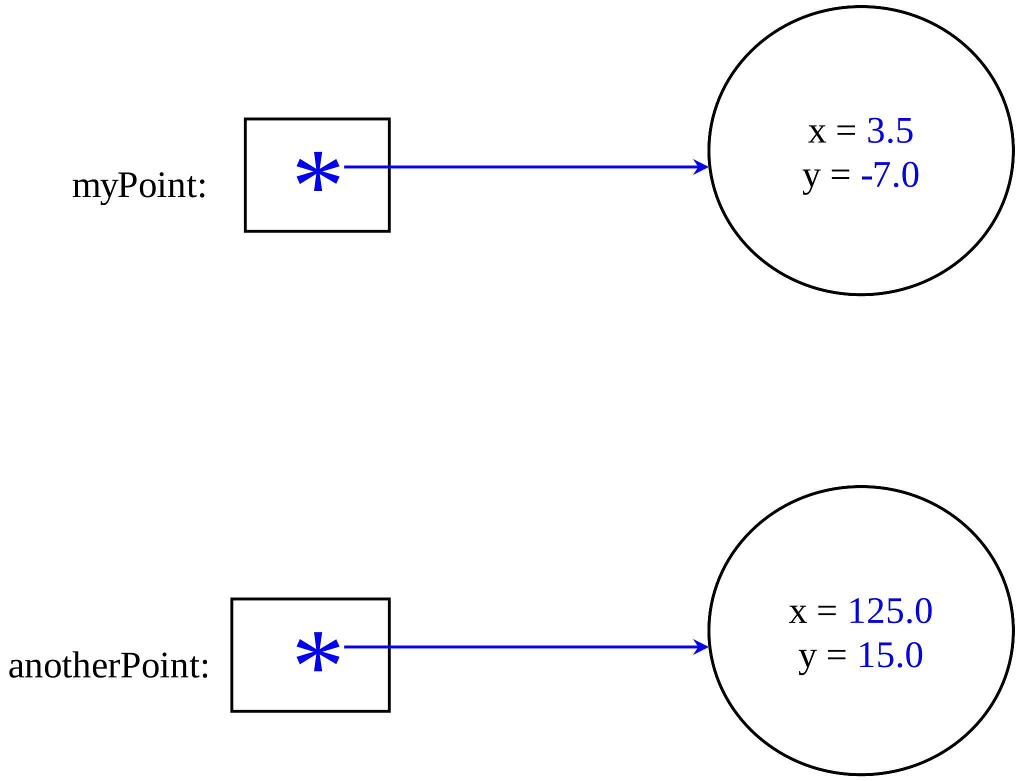
```
public class Point
{
        private double x, y;   ← instance variables

        public void setx(double value) {
                x = value;
        }

        public void sety(double value) {
                y = value;
        }
}
```

instance methods →

Data Members and Methods (continued)

```
            .
            .
Point myPoint = new Point();
myPoint.setx(3.5);
myPoint.sety(-7.0);

Point anotherPoint = new Point();
anotherPoint.setx(125.0);
anotherPoint.sety(15.0);
            .
            .
```

Fields and Methods (continued)



Class Data Members and Methods

- Example:

```
Public class Point
{
        private double x, y;
        private static int classID = 0;

        public void setx(double value) {
                x = value;
        }
        public void sety(double value) {
                y = value;
        }

        public static int classID() {
                return classID;
        }
}
```

class variable

class method

Calling Class Methods

invocation of a
class method

class name

```
int temp = Point.classID();
System.out.println("Point class id is " + temp);
```

Creating Objects

- Objects of a class are always instantiated by using the *new* operator.
- An *object reference* is a variable which "points to" the newly allocated object.
  - The object reference occupies stack memory.
  - The object reference can be changed so it points to some other object.
- The actual object occupies heap memory.

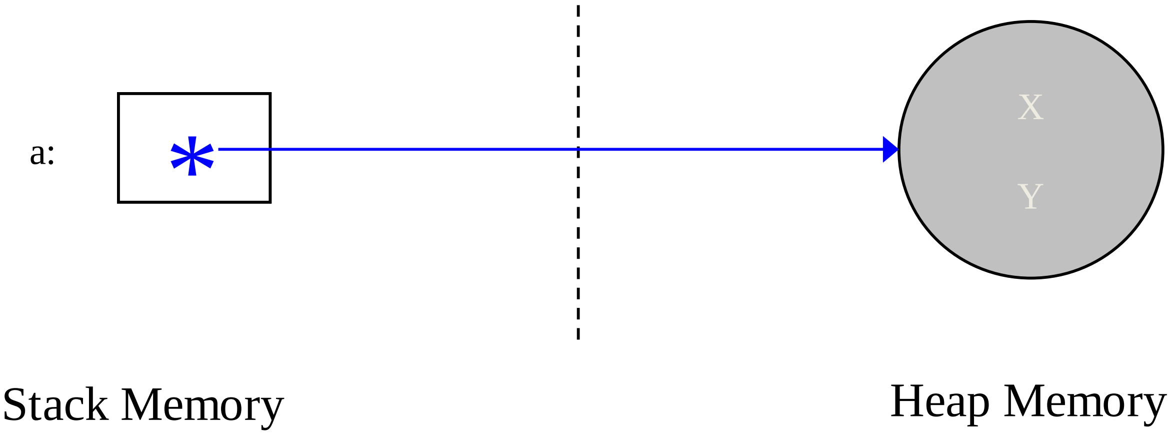
Creating Objects (continued)

- Example:

```
Point a = new Point();

Or:

Point b;

b = new Point();
```

creates an object reference on the stack

allocates memory for instance variables on the heap, and initializes their values



Constructors

- Like C++, normally constructors are used to initialize the data members of a newly created object.
- Like C++, Java constructors:
  - Have no return type.
  - Must have exactly the same name as the class.

Constructors -Example

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

    public Point(double xVal, double yVal)
    {
        x = xVal;
        y = yVal;
    }
}
```

constructor

Constructors (continued)

- Like C++, if you don't supply a constructor, a default constructor is provided automatically which is equivalent to:

```
public class Point {
    public Point() {
    }
}
```

Access Control

- Like C++ access from *other* classes is controlled by using the following access control keywords:
  - *public*:   accessible wherever the class is visible.
  - *protected*: accessible only to the class and its subclasses.
  - *private*: hidden from all other classes.

Access Control (continued)

- *Any method or variable that does not use one of the above modifiers has* *package* visibility.
  - It is visible to all classes within the same package.
  - If you do not explicitly declare a class to belong to a package, it is automatically put into the “default” package.

Access Control (continued)

- A protected method or variable is also accessible to all classes within the same package.
- In general, declare variables private or protected.
- In general, declare methods public or protected.
  - Occasionally, private methods are appropriate.
- In general, declare classes public.
  - Occasionally, more limited visibility for classes is appropriate.

More on Constructors

- Like C++, you can overload a constructor, and methods, as long as their signatures are different.
- One constructor can invoke another by using the this() statement *before* any other code.

More on Constructors (continued)

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

    public Point(double xVal, double yVal) {
        x = xVal;
        y = yVal;
    }

    public Point() {
        this(0.0, 0.0);
    }
}
```

invokes the above constructor, supplying default values for x and y

The "this" Keyword

- "this" can be also used for object self reference, or to invoke its own methods:

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

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

     public void setxy(double xVal, yVal) {
      this.setx(xVal);
      this.sety(yVal);
    }
}
```

self reference is necessary to distinguish the instance variables from the parameters

invokes the above two methods