Data Structures and Algorithms in Java

Object-oriented Programming: Defining Data Types

Outline

Program Template

Program Template

</> Program.java

```
// Package statement.
[package dsa;]

// Import statements.
...

// Outer class definition.
public class Program [implements <name>] {
    // Class/instance variable declarations.
    ...

    // Constructor definitions.
    ...

    // Method definitions.
    ...

    // Inner class definitions.
    ...

    // Function definitions.
    ...
}
```

Class Definitions · Outer

```
public class <program_name> [implements <name>] {
    // Class/instance variable declarations.
    ...

    // Constructor definitions.
    ...

    // Method definitions.
    ...

    // Inner class definitions.
    ...

    // Function definitions.
    ...
}
```

Class Definitions · Inner

```
private [static] class <name> [implements <name>] {
    // Class/instance variable declarations.
    ...

    // Constructor definitions.
    ...

    // Method definitions.
    ...

    // Function definitions.
    ...
}
```

Class/Instance Variable Declarations

Class/Instance Variable Declarations

Class variable declaration

```
public static <type> <name>;
```

Class/Instance Variable Declarations

Class variable declaration

```
public static <type> <name>;
```

A class variable is accessed as `[<class>].<name>`

Class/Instance Variable Declarations

Class variable declaration

```
public static <type> <name>;
```

A class variable is accessed as `[<class>].<name>`

Instance variable declaration

```
private <type> <name>;
```

Class/Instance Variable Declarations

Class variable declaration

```
public static <type> <name>;
```

A class variable is accessed as `[<class>].<name>`

Instance variable declaration

```
private <type> <name>;
```

An instance variable is accessed as `<object>.<name>`

Constructor Definitions

Constructor Definitions

```
public <class_name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

Constructor Definitions

```
public <class_name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

If a class has no explicit constructors, `javac` implicitly provides a zero-parameter constructor

Constructor Definitions

```
public <class_name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

If a class has no explicit constructors, `javac` implicitly provides a zero-parameter constructor

A constructor typically initializes the class/instance variables

Constructor Definitions

```
public <class_name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

If a class has no explicit constructors, `javac` implicitly provides a zero-parameter constructor

A constructor typically initializes the class/instance variables

Within a constructor, `this` is a reference to the object being constructed

Constructor Definitions

```
public <class_name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

If a class has no explicit constructors, `javac` implicitly provides a zero-parameter constructor

A constructor typically initializes the class/instance variables

Within a constructor, `this` is a reference to the object being constructed

A constructor can call another constructor within the same class as `this(<arg1>, <arg2>, ...)`

Method Definitions

Method Definitions

```
private|public <type>|void <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

Method Definitions

```
private|public <type>|void <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

A method is called as `<object>.<name>(<arg1>, <arg2>, ...)`

Method Definitions

```
private|public <type>|void <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

A method is called as `<object>.<name>(<arg1>, <arg2>, ...)`

Within a method, `this` is a reference to the object on which the method was called

Examples · `Stopwatch` Class (API)

`Stopwatch`	
`Stopwatch()`	constructs a stopwatch object
`double elapsedTime()`	returns the elapsed time (in seconds) since the creation of the stopwatch
`String toString()`	returns a string representation of the stopwatch

`TimeOps.java`

Command-line input	n (int)
Standard output	comparison of `Math.pow()` and `Math.sqrt()` from computing the sum $\sqrt{1} + \sqrt{2} + \cdots + \sqrt{n}$

Examples · `Stopwatch` Class (Application)

`TimeOps.java`

Command-line input	n (int)
Standard output	comparison of `Math.pow()` and `Math.sqrt()` from computing the sum $\sqrt{1} + \sqrt{2} + \cdots + \sqrt{n}$

`>_ ~/workspace/dsaj/programs`

```
$ _
```

`TimeOps.java`

Command-line input	n (int)
Standard output	comparison of `Math.pow()` and `Math.sqrt()` from computing the sum $\sqrt{1} + \sqrt{2} + \cdots + \sqrt{n}$

`~/workspace/dsaj/programs`

```
$ java TimeOps 10000000
```

Examples · `Stopwatch` Class (Application)

`TimeOps.java`

Command-line input	n (int)
Standard output	comparison of `Math.pow()` and `Math.sqrt()` from computing the sum $\sqrt{1} + \sqrt{2} + \cdots + \sqrt{n}$

`~/workspace/dsaj/programs`

```
$ java TimeOps 100000000
Math.sqrt() is 1.00 times faster than Math.pow()
$ _
```

Examples · Stopwatch Class (Application)

</> TimeOps.java

```
1  import stdlib.StdOut;
2
3  public class TimeOps {
4      public static void main(String[] args) {
5          int n = Integer.parseInt(args[0]);
6          Stopwatch watch1 = new Stopwatch();
7          double total = 0.0;
8          for (int i = 1; i <= n; i++) {
9              total += Math.sqrt(i);
10         }
11         double time1 = watch1.elapsedTime();
12         Stopwatch watch2 = new Stopwatch();
13         total = 0.0;
14         for (int i = 1; i <= n; i++) {
15             total += Math.pow(i, 0.5);
16         }
17         double time2 = watch2.elapsedTime();
18         StdOut.printf("Math.sqrt() is %.2f times faster than Math.pow()\n", time2 / time1);
19     }
20 }
```

Examples · `Stopwatch` Class (Implementation)

Stopwatch	
`Stopwatch()`	constructs a stopwatch object
`double elapsedTime()`	returns the elapsed time (in seconds) since the creation of the stopwatch
`String toString()`	returns a string representation of the stopwatch

Examples · `Stopwatch` Class (Implementation)

`Stopwatch`	
`Stopwatch()`	constructs a stopwatch object
`double elapsedTime()`	returns the elapsed time (in seconds) since the creation of the stopwatch
`String toString()`	returns a string representation of the stopwatch

Instance variables

- Creation time of the stopwatch, `long creationTime`

Examples · Stopwatch Class (Implementation)

Stopwatch.java

```java
import stdlib.StdOut;

public class Stopwatch {
    private long creationTime;

    public Stopwatch() {
        this.creationTime = System.currentTimeMillis();
    }

    public double elapsedTime() {
        return (System.currentTimeMillis() - this.creationTime) / 1000.0;
    }

    public String toString() {
        return "" + this.creationTime;
    }

    public static void main(String[] args) {
        // Unit tests the data type
    }
}
```

Examples · `Turtle` Class (API)

`Turtle`	
`Turtle(double x, double y, double theta)`	constructs a turtle object
`void turnLeft(double theta)`	rotates the turtle by `theta` degrees ccw
`void goForward(double stepSize)`	moves the turtle forward by distance `stepSize`, with pen down
`String toString()`	returns a string representation of the turtle

`Drunks.java`

Command-line input	*n* (int), *steps* (int), and *stepSize* (double)
Standard draw output	the paths of *n* turtles, each involving *steps* random steps of length *stepSize*

Examples · `Turtle` Class (Application)

Drunks.java

Command-line input	*n* (int), *steps* (int), and *stepSize* (double)
Standard draw output	the paths of *n* turtles, each involving *steps* random steps of length *stepSize*

```
>_ ~/workspace/dsaj/programs

$ _
```

Drunks.java

Command-line input	*n* (int), *steps* (int), and *stepSize* (double)
Standard draw output	the paths of *n* turtles, each involving *steps* random steps of length *stepSize*

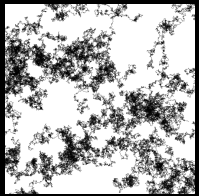
~/workspace/dsaj/programs

```
$ java Drunks 20 5000 0.005
```

Drunks.java

Command-line input	*n* (int), *steps* (int), and *stepSize* (double)
Standard draw output	the paths of *n* turtles, each involving *steps* random steps of length *stepSize*

```
>_ ~/workspace/dsaj/programs
$ java Drunks 20 5000 0.005
```



Drunks.java

Command-line input	*n* (int), *steps* (int), and *stepSize* (double)
Standard draw output	the paths of *n* turtles, each involving *steps* random steps of length *stepSize*

~/workspace/dsaj/programs

```
$ java Drunks 20 5000 0.005
$ _
```

Examples · Turtle Class (Application)

</> Drunks.java

```java
import stdlib.StdDraw;
import stdlib.StdRandom;

public class Drunks {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int steps = Integer.parseInt(args[1]);
        double stepSize = Double.parseDouble(args[2]);
        Turtle[] turtles = new Turtle[n];
        for (int i = 0; i < n; i++) {
            double x = StdRandom.uniform(0.0, 1.0);
            double y = StdRandom.uniform(0.0, 1.0);
            double theta = StdRandom.uniform(0.0, 360.0);
            turtles[i] = new Turtle(x, y, theta);
        }
        StdDraw.setPenRadius(0.001);
        StdDraw.enableDoubleBuffering();
        for (int i = 0; i < steps; i++) {
            for (int j = 0; j < turtles.length; j++) {
                Turtle turtle = turtles[j];
                double theta = StdRandom.uniform(0.0, 360.0);
                turtle.turnLeft(theta);
                turtle.goForward(stepSize);
                StdDraw.show();
            }
        }
    }
}
```

`Turtle`	
`Turtle(double x, double y, double theta)`	constructs a turtle object
`void turnLeft(double theta)`	rotates the turtle by `theta` degrees ccw
`void goForward(double stepSize)`	moves the turtle forward by distance `stepSize`, with pen down
`String toString()`	returns a string representation of the turtle

Examples · `Turtle` Class (Implementation)

`Turtle`	
`Turtle(double x, double y, double theta)`	constructs a turtle object
`void turnLeft(double theta)`	rotates the turtle by `theta` degrees ccw
`void goForward(double stepSize)`	moves the turtle forward by distance `stepSize`, with pen down
`String toString()`	returns a string representation of the turtle

Instance variables

- x-coordinate of the turtle, `double x`
- y-coordinate of the turtle, `double y`
- ccw angle (in degrees) of the turtle, `double theta`

Examples · Turtle Class (Implementation)

</> Turtle.java

```java
import stdlib.StdDraw;

public class Turtle {
    private double x;
    private double y;
    private double theta;

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

    public void turnLeft(double theta) {
        this.theta += theta;
    }

    public void goForward(double stepSize) {
        double xOld = this.x;
        double yOld = this.y;
        this.x += stepSize * Math.cos(Math.toRadians(this.theta));
        this.y += stepSize * Math.sin(Math.toRadians(this.theta));
        StdDraw.line(xOld, yOld, this.x, this.y);
    }

    public String toString() {
        return "(" + this.x + ", " + this.y + ", " + this.theta + ")";
    }

    public static void main(String[] args) {
        // Unit tests the data type
    }
}
```