6.092 Introduction to Software Engineering in Java
January (IAP) 2009

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.
6.092: Intro to Java

4: Classes, Objects
Assignment 3

A group of friends participate in the Boston Marathon.

- Find the best performer.
- Find the second-best performer.
public static int getMinIndex(int[] values) {

    int minValue = Integer.MAX_VALUE;
    int minIndex = -1;

    for(int i=0; i<values.length; i++)
        if(values[i] < minValue) {
            minValue = values[i];
            minIndex = i;
        }

    return minIndex;
}
public static int getSecondMinIndex(int[] values) {
    int secondMinValue = Integer.MAX_VALUE;
    int secondMinIndex = -1;
    int minIndex = getMinIndex(values);

    for(int i=0; i<values.length; i++) {
        if(i == minIndex)
            continue;
        if(values[i] < secondMinValue) {
            secondMinValue = values[i];
            secondMinIndex = i;
        }
    }
    return secondMinIndex;
}
Frequent Issues (I)

Array index VS array values

for(int i=0; i< array.length; i++) {
    System.out.println(i) // index
    System.out.println(array[i]) // value @ i
}

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>-300</td>
<td>0</td>
<td>42</td>
<td></td>
</tr>
</tbody>
</table>
Frequent Issues (II)

Curly braces \{ ... \} in loops

- Without braces, only one statement is in the loop

```java
int i=0;
for(i=0; i<10; i++)
    System.out.println(“Inside loop”); // executed 10x
System.out.println(i); // executed 1x only

for(i=0; i<10; i++) {
    System.out.println(“Inside loop”); // executed 10x
    System.out.println(i); // executed 10x
}
```

- Similarly for if / else blocks...
Frequent Issues (III)

Be careful when initializing variables

```java
public static int getMin(int[] values) {
    int minValue;
    minValue = 0; // X
    minValue = 1000; // ✓ if ∃ value<=1000
    minValue = values[0]; // ✓
    minValue = Integer.MAX_VALUE; // ✓

    for(int i=0; i< values.length; i++)
        if(values[i] < minValue)
            minValue = values[i];

    return minValue;
}
```
Frequent Issues (IV)

Second min & variable initialization

```java
public static int getSecondMin(int[] values) {
    int min = getMin(values);
    int secondMin;
    if(values[0] == min)
        secondMin = values[1];
    else
        secondMin = values[0];

    for(int i=0; i<values.length; i++) {
        if(values[i] == min)
            continue;
        if(values[i] < secondMin)
            secondMin = values[i];
    }
    return secondMin;
}
```
Building & Debugging Programs

• Write programs block by block

• Use printlns to ensure that your block is correct

```java
for(int i=0; i< values.length; i++)
    if(values[i] < minValue) {
        System.out.println("Current min: "+ minValue);
        System.out.println("New min: "+ values[i]);
        minValue = values[i];
    }

System.out.println("Final min: "+ minValue);
```
What we have learned so far...

- Variables & types
- Operators
- Type conversions & casting
- Methods & parameters
- *If* statement
- Loops
- Arrays

What we are going to learn today...

- Classes & Objects
  *(object-oriented programming OOP)*
Object-Oriented Programming

How do you represent the world on your computer?
Object-Oriented Programming

int, float, double, strings are low-level.

Can we do things at a higher level?
Objects

Objects are a collection of related data and methods.
Objects

Example: Strings

A string is a collection of characters (letters) and has a set of methods built in.

```java
String nextTrip = "Mexico";
int size = nextTrip.length(); // 6
```
Objects

To create a new object, use the `new` operator. If you do not use `new`, you are making a reference to an object (i.e. a pointer to the same object).
Objects

Point p;
p.x = 23;
p.y = -12;

public static Point middlePoint (Point p1, Point p2) {
    Point q = new Point ((p1.x+p2.x)/2, (p1.y+p2.y)/2);
    return q;
}
Objects and references

Point p1 = new Point (12,34);
Point p2 = p1;

System.out.println("x = " + p2.x); // 12
p1.x = 24;
System.out.println("x = " + p2.x); // 24!
Objects and references

Point p1 = new Point(12, 34);

Point p1

Point p2 = p1;

Point p2

... 12 ... 34 ...
null simply represents no object. It is convenient as a return value to mean that a method failed for example. It may also be used to “remove” an element from an array.

```java
String values[] = { "Adam", "Bob", "Mary" };
values[1] = null;
```
Classes

A class is a prototype to design objects.

It is a set of variables and methods that encapsulates the properties of the class of objects.
Classes

In Java, you can (will) create several classes in a project.
Syntax Issues

- Class names begin with a capital letter (e.g. WeeklyPay)
- One class = one file
- Only one class has the main method (startup class)
```java
class Bicycle {
    int speed;
    int gear;

    void changeGear () {
        gear += 1;
    }

    void speedUp () {
        speed += 10;
    }

    void slowDown () {
        speed -= 10;
    }
}
```
Here we create two separate objects of the class Bicycle.

```java
class CambridgeBicycleStore {

    public static void main (String arguments[]) {
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();
        bike1.speedUp();
    }
}
```
Class constructors

A class constructor is called each time an object of the class is *instantiated* (created).
class Bicycle {
    public int speed;
    public int gear;

    public Bicycle() {
        speed = 0;
        gear = 1;
    }
}
public static void main (String[] arguments) {
    Bicycle bike = new Bicycle ();
    System.out.println (bike.speed); // 0
    System.out.println (bike.gear); // 1
}
Constructors with arguments

class Bicycle {
    int speed;
    int gear;

    public Bicycle(int speedval, int gearval) {
        speed = speedval;
        gear = gearval;
    }
}

Constructors with arguments

```java
public static void main (String[] arguments) {
    Bicycle bike = new Bike (10,3);
    System.out.println (bike.speed); // 10
    System.out.println (bike.gear); // 3
}
```
Reference vs primitive

- **Primitive** types are the basic types of data
  - byte, short, int, long, float, double, boolean, char
  - primitive variables store primitive values

- **Reference** types are any instantiable class as well as arrays
  - String, Scanner, Random, Point, int[], String[], etc.
  - reference variables store addresses

```java
int var1 = 12;
NewC var2 = new NewC(12);
```
Reference vs primitive

class Example {
    public static void compute (int x) {
        x = 2;
    }

    public static void main(String[] arguments) {
        int y = 1;
        System.out.println(y); // 1
        compute (y);
        System.out.println(y); // 1
    }
}
class Example {
    public static void compute (Point p) {
        p.x = 2;
    }

    public static void main(String[] arguments) {
        Point q = new Point(1,1);
        System.out.println(q.x); // 1
        compute (q);
        System.out.println(q.x); // 2!
    }
}
Reference vs primitive

```
int y
1
compute(y)
...
int x
2
```

```
Point q
.
compute(q)
...
Point p
.
```
Object methods vs class methods

So far, we have seen class methods.

Class methods are declared static.
class Bicycle {
    static int gear, speed;

    public static void speedUp (Bicycle b) {
        b.speed += 10;
    }

    public static void main (String[] arguments) {
        Bicycle bike1 = new Bicycle();
        speedUp (bike1);
    }
}

static means that the variable is going to be same for all objects of the class.
class Bicycle {
    static int gear, speed;

    public static void speedUp (Bicycle b) {
        b.speed += 10;
    }

    public static void main (String[] arguments) {
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();
        System.out.println(bike2.speed); // 0
        speedUp (bike1);
        System.out.println(bike2.speed); // 10!
    }
}

This is not convenient!!
We changed the speed on bike 1 and it automatically changed the speed on bike 2!!
class Bicycle {
    static int gear, speed;

    public static void speedUp (Bicycle b) {
        b.speed += 10;
    }

    public static void main (String[] arguments) {
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        System.out.println(bike2.speed); // 0
        speedUp (bike1);
        System.out.println(bike2.speed); // 0
    }
}

non-static means that the variable is going to be different for each object.
Object methods vs class methods

The same concept applies to methods.

**static** methods are defined for a class.

**non-static** methods are defined for an object.
class Bicycle {
    static int gear, speed;

    public static void speedUp () {
        speed += 10;
    }

    public static void main (String[] arguments) {
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        System.out.println(bike2.speed); // 0
        bike1.speedUp ();
        System.out.println(bike2.speed); // 0
    }
}
static or non-static?

• As you like!
• Most of the time, both will work.

• However, one may make more sense than the other:

  speedUp should be non-static (object)
  comparePrice(bike1, bike2) should be static (class)
static (class)

```java
static void speedUp (Bicycle b); // declare
speedUp (bike1); // call
```

---

non-static (object)

```java
void speedUp(); // declare
bike1.speedUp(); // call
```
A common mistake

“non-static variable gear cannot be referenced from a static context“

class Bicycle {
    int gear, speed;

    public static void speedUp () {
        speed += 10; // WRONG!!
    }
}
Packages

A package is a set of classes that relate to a same purpose.

Example: math, graphics, input/output...

In order to use a package, you have to import it.
Assignment 4

Electronic system to borrow / return books

• Book class

• Library class