Creating Your Own Classes

"Code with Passion!"

Topics

- Defining your own classes
- Instance variables vs. Static variables
- Methods
  - Instance methods
  - Static methods
- Overloading methods
- Constructors
- “this” reference
- Access modifiers

Defining Your Own Class

Defining your own classes

- Class definition structure looks like following

```
<access modifier> class <name> {
        <attributeDeclaration>*
        <constructorDeclaration>*
        <methodDeclaration>*
}
```

Example: Define StudentRecord Class

```
public class StudentRecord {
      //we'll add more code here later
}
```

- public – access modifier
- class - this is the keyword you must use
- StudentRecord – class name of your choice

Coding Guidelines

- Give a meaningful name to your class
- Class name typically starts with a Capital letter – not a requirement, however
- The file name of your class must have the SAME NAME as your class name
  - StudentRecord class should be defined in StudentRecord.java
- You can have multiple class declarations in a single file, however, if there is a class declaration that matches the file name
  - When Class A is used only with Class B, you can declare both Class A and Class B in the B.java source file - You might declare class A with “private class A {..}” so that it can be accessible only by B in this case

Lab:

Exercise 1: Create your own class
1014_javase_createclass.zip

Instance Variables vs. Static Variables

Instance Variables (Properties) vs. Class (Static) Variables

- Instance Variables
  - Belongs to an object instance
  - Values of variables of an object instance are different from the ones of other object object instances from the same class

- Class Variables (also called static variables)
  - Variables that belong to the class.
  - This means that they have the same value for all the object instances in the same class

Class Variables

Car Class		Object Car A	Object Car B
Instance Variables	Plate Number	ABC 111	XYZ 123
	Color	Blue	Red
	Manufacturer	Mitsubishi	Toyota
	Current Speed	50 km/h	100 km/h
Class Variable	Count = 2		
Instance Methods	Accelerate Method		
	Turn Method		
	Brake Method		

Instance Variables

Declaring Properties (Attributes)

- To declare a certain attribute for our class, we write,

```
<access modifier> <type> <name> [=
  <default_value>];
```

Instance Variables

```
public class StudentRecord {

   // Instance variables
   private String     name;
   private String     address;
   private int   age;
   private double     mathGrade;
   private double     englishGrade;
   private double     scienceGrade;
   private double     average;

   //we'll add more code here later
}
```

- private is access modifier

Static Variables

Static (Class) variables

```
public class StudentRecord {
// static variables
private static int studentCount;
// we'll add more code here later
}
```

- we use the keyword static to indicate that a variable is a static (class) variable.

Methods

Declaring Methods

- To declare a method, you write,

```
<access modifier> <returnType>
  <name>(<parameter>*){
        <statement>*
}
```

  - <modifier> can carry a number of different modifiers
  - <returnType> can be any data type (including void)
  - <name> can be any valid identifier
  - <parameter> ::= <parameter_type> <parameter_name>[,]

Accessor (Getter) Methods

- Used to get (retrieve) data (in the form of object or primitive) from our class variables (instance/static).
- Written as:

```
get<NameOfInstanceVariable>
```

Example 1: Accessor (Getter) Method

```
public class StudentRecord {
private String name;
:
public String getName(){
return name;
}
}
```

- public - means that the method can be called from objects of other classes
- String - is the return type of the method. This means that the method should return an object of type String
- getName - the name of the method
- () - this means that our method does not have any parameters

Example 2: Accessor (Getter) Method

```
public class StudentRecord {
  private String   name;
  // some code

  // An example in which the business logic is
  // used to return a value on an accessor method
  public double getAverage(){
    double result = 0;
    result=(mathGrade+englishGrade+scienceGrade)/3;
    return result;
  }
}
```

Mutator (Setter) Methods

- Used to write or change values of a variable
- Written as:

```
set<NameOfInstanceVariable>
```

Example: Mutator (Setter) Method

```
public class StudentRecord {
private String name;
:
public void setName( String arg ){
name = arg;
}
}
```

- public - “public” access modifier
- void - means that the method does not return any value
- setName - the name of the method
- (String arg) - argument that will be used inside our method

Multiple return statements

- You can have multiple return statements for a method as long as they are not on the same block

```
public String getNumberInWords( int num ){
String defaultNum = "zero";
if( num == 1 ){
  return "one";
}
else if( num == 2){
  return "two";
}
  return defaultNum;
}
```

Static Methods

Static methods

```
public class StudentRecord {
    private static int studentCount;
    public static int getStudentCount(){
     return studentCount;
    }
}
```

- public - “public” access modifier
- static - means that the method is static
- int - is the return type of the method. This means that the method should return a value of type int
- getStudentCount - the name of the method
- () - this means that our method does not have any parameters

Coding Guidelines for Methods

- Method names should start with a small letter
- Method names should sound like a verb
- Provide documentation before the declaration of the method. You can use Javadocs style for this.

Source Code for StudentRecord class

```
public class StudentRecord {

   // Instance variables
   private String      name;
   private String      address;
   private int         age;
   private double      mathGrade;
   private double      englishGrade;
   private double      scienceGrade;
   private double      average;
   private static int studentCount;
```

Source Code for StudentRecord Class

```
/**
 * Returns the name of the student (Accessor method)
 */
public String getName(){
   return name;
}

/**
 * Changes the name of the student (Mutator method)
 */
public void setName( String temp ){
   name = temp;
}
```

Source Code for StudentRecord Class

```
/**
 * Computes the average of the english, math and science
 * grades (Accessor method)
 */
public double getAverage(){
    double result = 0;
    result = ( mathGrade+englishGrade+scienceGrade )/3;
    return result;
}
/**
 * returns the number of instances of StudentRecords
 * (Accessor method)
 */
public static int getStudentCount(){
    return studentCount;
}
```

Sample Source Code that uses StudentRecord Class

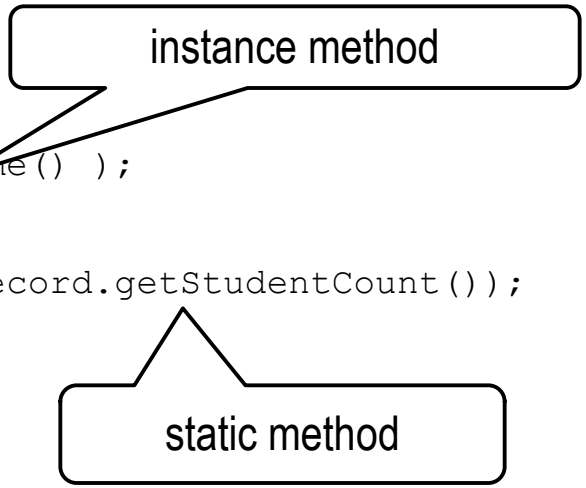
```
public class StudentRecordExample
{
   public static void main( String[] args ){

    //create three objects for Student record
    StudentRecord   annaRecord = new StudentRecord();
    StudentRecord   beahRecord = new StudentRecord();
    StudentRecord   crisRecord = new StudentRecord();

    //set the name of the students
    annaRecord.setName("Anna");
    beahRecord.setName("Beah");
    crisRecord.setName("Cris");

    //print anna's name
    System.out.println( annaRecord.getName() );

    //print number of students
    System.out.println("Count="+StudentRecord.getStudentCount());
   }
}
```



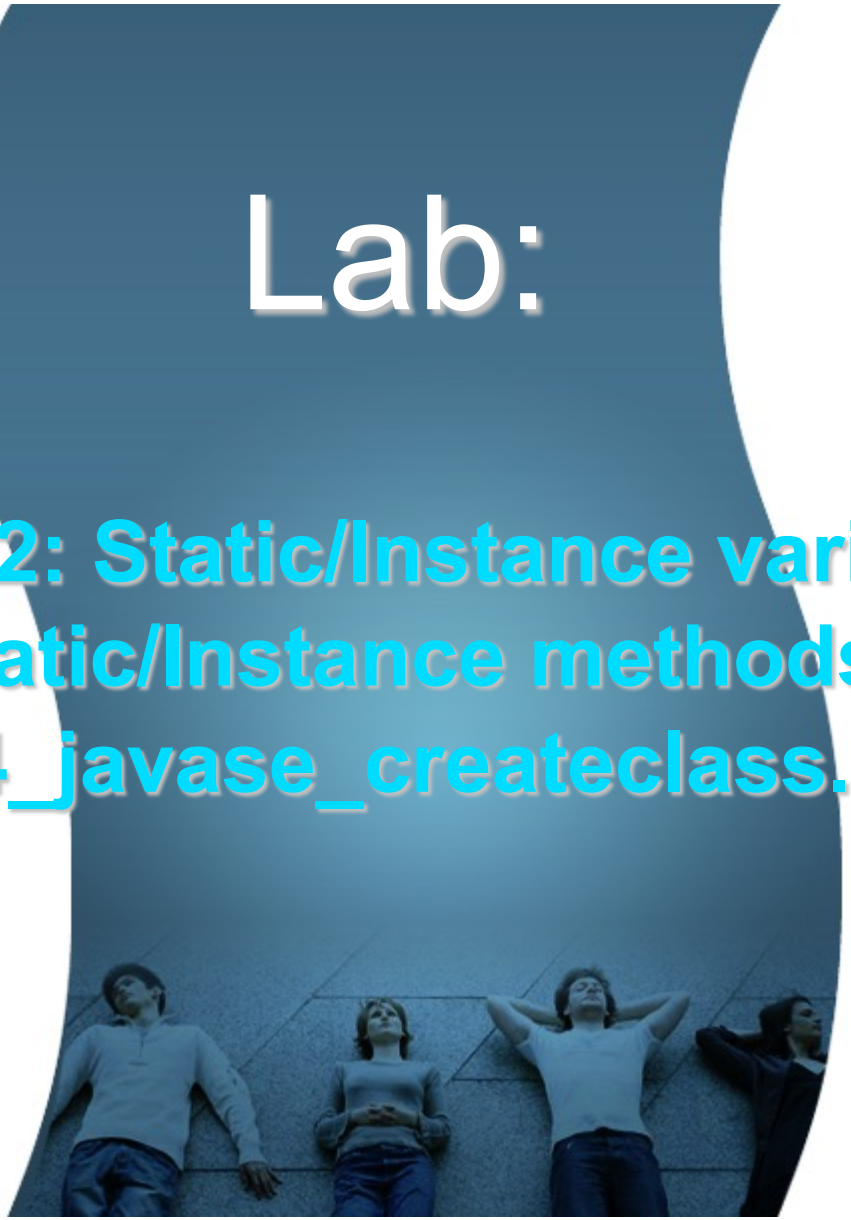
When do you want create Static Methods?

- When the logic and state does not involve specific object instance
  - Computation method, for example
  - Computer.add(int x, int y); // add(..) is a static method

- When the logic is a convenience without creating an object instance
  - Integer.parseInt(“34”); // parseInt(..) is a static method

Lab:

Exercise 2: Static/Instance variables & Static/Instance methods
1014_javase_createclass.zip



Overloading Methods

Method Overloading

- What is Method overloading?
  - Methods with the same names but different number of arguments or different types of arguments can coexist in a single class
- Why Method overloading?
  - To allow different implementations (but similar behavior) with a same method name but with different number of parameters or different types of parameters
- Always remember that overloaded methods have the following properties:
  - The same method name
  - Different number of parameters or different types of parameters
  - Return types can be different or the same, however

Example: Overloaded Methods

```
public void print( String temp ){
    System.out.println("Name:" + name);
    System.out.println("Address:" + address);
    System.out.println("Age:" + age);
}

public void print(double eGrade, double mGrade,
                  double sGrade)
    System.out.println("Name:" + name);
    System.out.println("Math Grade:" + mGrade);
    System.out.println("English Grade:" + eGrade);
    System.out.println("Science Grade:" + sGrade);
}
```

Example: Calling Overloaded Methods

```
public static void main( String[] args )
{
    StudentRecord  annaRecord = new StudentRecord();

    annaRecord.setName("Anna");
    annaRecord.setAddress("Philippines");
    annaRecord.setAge(15);
    annaRecord.setMathGrade(80);
    annaRecord.setEnglishGrade(95.5);
    annaRecord.setScienceGrade(100);

    //overloaded methods
    annaRecord.print( annaRecord.getName() );
    annaRecord.print( annaRecord.getEnglishGrade(),
              annaRecord.getMathGrade(),
                  annaRecord.getScienceGrade());
}
```

Lab:

Exercise 3: Overloading Methods
1014_javase_createclass.zip

Constructors (Constructor Methods)

Constructors

- It is a special method where initialization of the newly created object is done

- Differences from a regular method
  - Constructors have the same name as the class
  - Constructors does not have any return value
  - You cannot call a constructor directly, it gets called indirectly when object gets instantiated

Constructors

- To declare a constructor, we write,

```
<modifier> <className> (<parameter>*) {
        <statement>*
}
```

Default Constructor (Method)

- Is the constructor with no arguments.
  - Also called no-arg constructor
- If the class does not specify any constructors, then an default constructor gets created automatically by the compiler

```
// Default constructor of StudentRecord class
public StudentRecord(){
}
```

- If there is already a constructor, then default constructor does not get created automatically by the compiler

Overloading Constructor Methods

```
public StudentRecord(){
    //some initialization code here
}
public StudentRecord(String temp){
    this.name = temp;
}
public StudentRecord(String name, String address){
    this.name = name;
    this.address = address;
}
public StudentRecord(double mGrade, double eGrade,
                     double sGrade){
    mathGrade = mGrade;
    englishGrade = eGrade;
    scienceGrade = sGrade;
}
```

Example: Using Constructors

```
public static void main( String[] args ){
    //create three objects for Student record
    StudentRecord     annaRecord = new StudentRecord("Anna");
    StudentRecord     beahRecord =
                          new StudentRecord("Beah",
                                            "Philippines");
    StudentRecord     crisRecord=
                          new StudentRecord(80,90,100);
    //some code here
}
```

“this(..)” constructor call

- Constructor calls can be chained, meaning, you can call another constructor from inside a constructor
  - Use this(..) call for that purpose

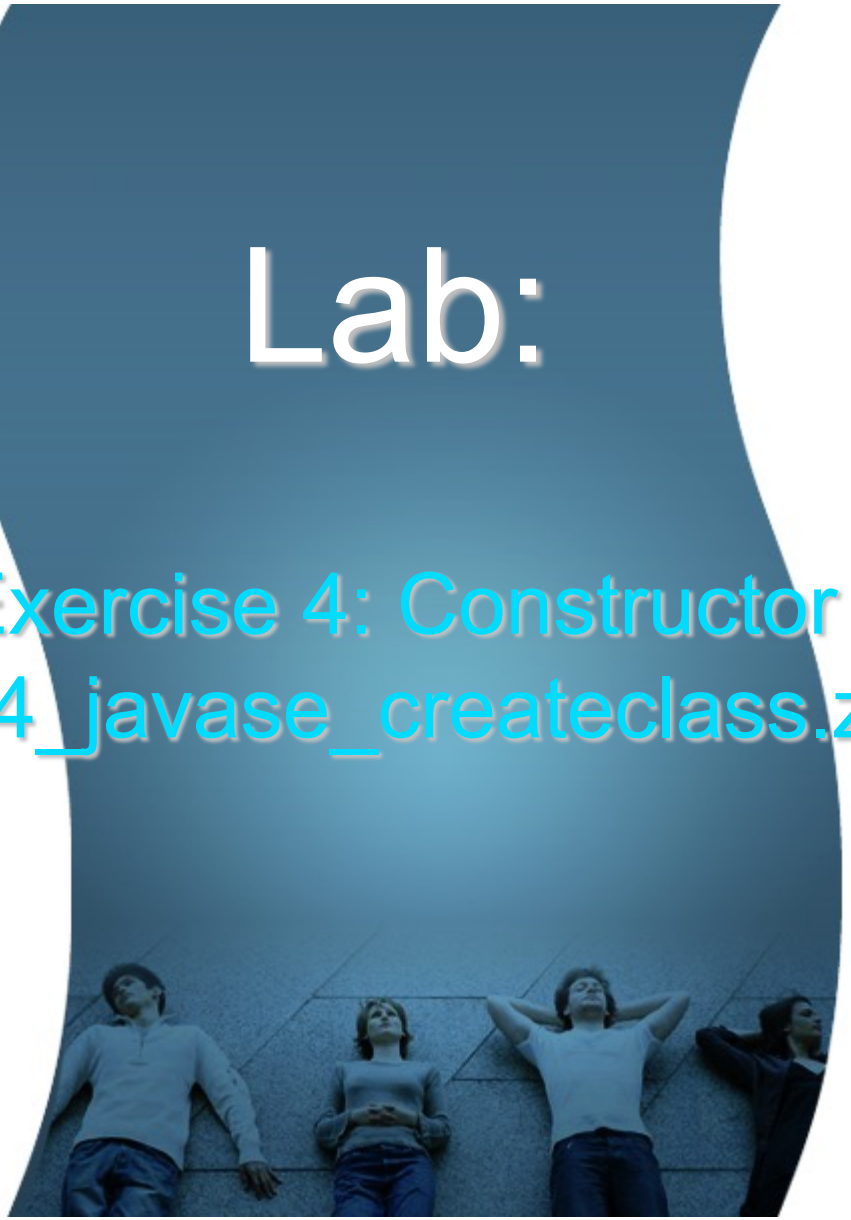
- There are a few things to remember when using the this(..) constructor call:
  - It can only be used in a constructor
  - It must occur at the first statement in a constructor

Example: this(..) constructor call

```
1: public StudentRecord(){
2:     this("some string");
3:
4: }
5:
6: public StudentRecord(String temp){
7:     this.name = temp;
8: }
9:
10: public static void main( String[] args )
11: {
12:
13:    StudentRecord    annaRecord = new StudentRecord();
14: }
```

Lab:

Exercise 4: Constructor
1014_javase_createclass.zip



“this” Reference

“this” reference

- The *this* reference
  - Refers to current object instance itself
  - Used to access the instance variables
- To use the this reference, we type,
  `this.<nameOfTheInstanceVariable>`

```
int age;
public void setAge( int age ){
    this.age = age;  // same as age = age;
 }
```

- You can only use the this reference for instance variables and NOT static variables
  - Because *this* refers to an object instance

“this” reference

- The *this* reference is assumed when you call a method from the same object

```
// MyClass definition
public class MyClass {

    void aMethod() {
        // same thing as this.anotherMethod()
        anotherMethod();
    }

    void anotherMethod() {
    // method definition here...
    }
}
```

Lab:

Exercise 5: “this”
1014_javase_createclass.zip

Access Modifiers

4 Types of Access Modifiers

- There are four different types of member access modifiers
  - public (Least restrictive)
  - protected
  - default (no access modifier is specified - package-private)
  - private (Most restrictive)

- The three access modifiers in blue color are explicitly written in the code to indicate the access type, for the 3rd one ("default"), no access modifier is specified

“public” Access Modifier

- Specifies that class members (variables or methods) are accessible to anyone, inside and outside the class and outside of the package to which the class belongs

```
public class StudentRecord {
    // public access to instance variable
    public int name;

    // public access to method
    public String getName(){
        return name;
    }
}
```

“protected” Access Modifier

- Specifies that the class members are accessible only to methods
  - in that class and
  - Classes in the same package
  - Subclasses of the class in different packages

```
public class StudentRecord {
    //protected access to instance variable
    protected String name;

    //protected access to method
    protected String getName(){
        return name;
    }
}
```

default Access Modifier

- Specifies that only classes in the same package can have access to the class' variables and methods
  - Sometimes called “Package-private”

```
public class StudentRecord {
  //default access to instance variable
  int    name;

  //default access to method
  String getName(){
    return name;
  }
}
```

“private” Access Modifier

- Specifies that the class members are only accessible within the class

```
public class StudentRecord {
    //private access to instance variable
    private int    name;

    //private access to method
    private String getName(){
      return name;
    }
}
```

Access Modifiers

	private	default/package	*protected*	*public*
Same class	Yes	Yes	Yes	Yes
Same package		Yes	Yes	Yes
Different package (subclass)			Yes	Yes
Different package (non-subclass)				Yes

Coding Guidelines

- The instance variables of a class should normally be declared private, and the class will just provide accessor and mutator methods to these variables.

Lab:

Exercise 6: Access Modifiers
1014_javase_createclass.zip

Code with Passion!