Creating Your Own Classes

Sang Shin
Michèle Garoche
www.javapassion.com
"Learn with Passion!"



Objectives

At the end of the lesson, the students should be able to:

- Create their own classes
- Declare properties (fields) and methods for their classes
- Use the this reference to access instance data
- Create and call overloaded methods
- Use access modifiers to control access to class members





Defining Your Own Class

Defining your own classes

Things to take note of for the syntax defined in this section:

```
or more occurrences of the line where it was applied to.
```

[] this part is optional



Defining your own classes

To define a class, we write:

- where
 - <modifier> is an access modifier, which may be combined with other types of modifier.



Example: Define StudentRecord Class

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

- where,
 - public means that our class is accessible to other classes outside the package
 - class this is the keyword used to create a class in Java
 - StudentRecord a unique identifier that describes our class



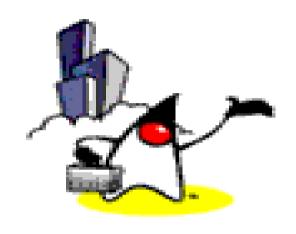
Coding Guidelines

- Think of an appropriate name for your class. Don't just call your class XYZ or any random names you can think of.
- Class names 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









Instance Variables Vs. Static Variables

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

- Instance Variables
 - Belongs to an object instance
 - Value of variable of an object instance is 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.

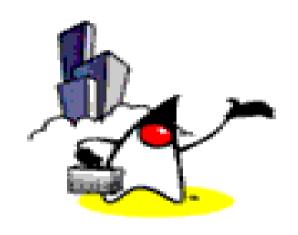


Class Variables

• For example,

		1 0 1 0 1 0 1 0 1		
Car Class	Object Car A	Object Car B		
Plate Number	ABC 111	XYZ 123		
Color	Blue	Red		
Manufacturer	Mitsubishi	Toyota		
Current Speed	50 km/h	100 km/h		
Count = 2				
}	Accelerate Method			
Accelerate Method Turn Method Brake Method				
	Brake Method			
	Plate Number Color Manufacturer Current Speed	Plate Number ABC 111 Color Blue Manufacturer Mitsubishi Current Speed 50 km/h Count = 2 Accelerate Method Turn Method		





Instance Variables

Declaring Properties (Attributes)

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



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
}
- where,
```

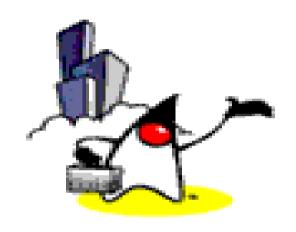
 private here means that the variables are only accessible within the class. Objects of other classes cannot access these variables directly. We will cover more about accessibility later.



Coding Guidelines for Instance Variables

- Declare all your instance variables right after "public class Myclass {"
- Declare one variable for each line.
- Instance variables, like any other variables should start with a small letter.
- Use an appropriate data type for each variable you declare. (mandatory)
- Declare instance variables as private so that only methods in the same class can access them directly.
 - Encapsulation





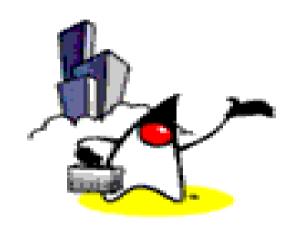
Static Variables

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 methods we write,

- <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

- Accessor methods
 - used to get (retrieve) data (in the form of object or primitive) from our class variables (instance/static).
 - usually written as:

get<NameOfInstanceVariable>



Example 1: Accessor (Getter) Method

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

- 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

- Mutator Methods
 - used to write or change values of a variable
 - Usually written as:

set<NameOfInstanceVariable>



Example: Mutator (Setter) Method

```
public class StudentRecord {
    private String name;
    :
    public void setName( String temp ) {
        name = temp;
    }
} - where,
```

- public means that the method can be called from objects of other classes
- void means that the method does not return any value
- setName the name of the method
- (String temp) parameter that will be used inside our method



Multiple return statements

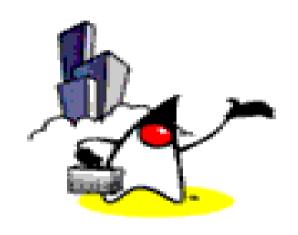
 You can have multiple return statements for a method as long as they are not on the same block



Example: Multiple return statements

```
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;
    }
}
- where,
```

- public- means that the method can be called from objects of other classes
- static-means that the method is static and should be called by typing, [ClassName]. [methodName]. For example, in this case, we call the method StudentRecord.getStudentCount()
- 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.



When to Define Static Method?

- When the logic and state does not involve object instances (different property values of different object instances do not play a role in the logic)
 - Computation method
 - add(int x, int y) method
- When the logic is a convenience without creating an object instance
 - int myint = Integer.parseInt("7");



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");
                                            instance method
   crisRecord.setName("Cris");
   //print anna's name
   System.out.println( annaRecord.getName() );
   //print number of students
   System.out.println("Count="+StudentRecord.getStudentCount());
                                            static method
```



Program Output

Anna

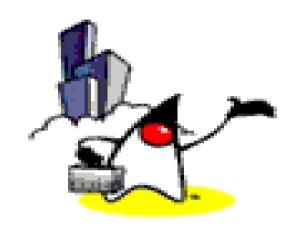
Student Count = 0



Demo:

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





Overloading Methods

Method Overloading

- Method overloading
 - allows a method with the same name but different parameters, to have different implementations and return values of different types
 - can be used when the same operation has different implementations.
- 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

Method Overloading Example



Example



Output

we will have the output for the first call to print,

Name: Anna

Address: Philippines

Age: 15

we will have the output for the second call to print,

Name: Anna

Math Grade:80.0

English Grade: 95.5

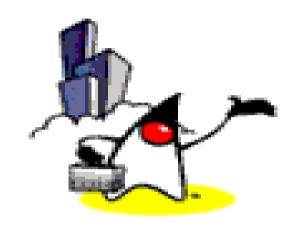
Science Grade: 100.0





Exercise 3: Overloading Methods 1014_javase_createclass.zip





Constructors (Constructor (Methods)

Constructors

- Constructors are important in instantiating an object. It is a method where all the initializations are placed.
- The following are the properties of a constructor:
 - Constructors have the same name as the class
 - A constructor is just like an ordinary method with some differences
 - 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,



Default Constructor (Method)

- The default constructor (no-arg constructor)
 - is the constructor without any parameters.
 - If the class does not specify any constructors, then an default constructor gets created automatically by the compiler
 - If there is already a constructor, then default constructor does not get created automatically by the compiler



Example: Default Constructor Method of StudentRecord Class

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



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 another constructor.
- We use the this() call for this
- There are a few things to remember when using the this() constructor call:
 - When using the this constructor call, IT MUST OCCUR AS THE FIRST STATEMENT in a constructor
 - It can ONLY BE USED IN A CONSTRUCTOR DEFINITION. The this call can then be followed by any other relevant statements.



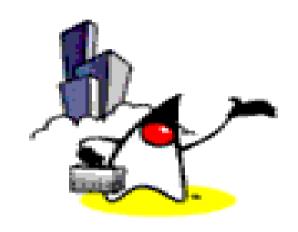
Example

```
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: }
```









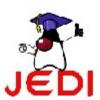
"this" Reference

"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>
```

- 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

```
public class MyClass {
    void aMethod() {
        // same thing as this.anotherMethod()
        anotherMethod();
    }
    void anotherMethod() {
        // method definition here...
    }
```

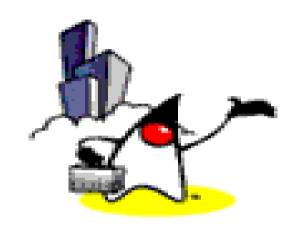


Example

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







Access Modifiers

Access Modifiers

- There are four different types of member access modifiers in Java:
 - public (Least restrictive)
 - protected
 - default
 - 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 keyword is used.



public accessibility

- public access
 - specifies that class members (variables or methods) are accessible to anyone, both inside and outside the class and outside of the package to which the class belongs
 - Any object that interacts with the class can have access to the public members of the class.
 - Keyword: public



Example: "public" Access Modifer

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

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



protected accessibility

- protected access
 - Specifies that the class members are accessible only to methods in that class and the subclasses of the class.
 - The subclass can be in different packages
 - Keyword: protected



Example: "protected" Access Modifier

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

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



default accessibility

Default access

- specifies that only classes in the same package can have access to the class' variables and methods
- no actual keyword for the default modifier; it is applied in the absence of an access modifier.



Example

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

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



private accessibility

- private accessibility
 - specifies that the class members are only accessible within the class
 - Keyword: private



Example: "private" Access Modifier

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

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



Java Program Structure: The 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.





Summary

- Defining your own classes
- Declaring Fields (instance, static/class)
- Declaring Methods (accessor, mutator, static)
- Returning values and Multiple return statements
- The this reference
- Method overloading
- Constructors (default, overloading, this() call)
- Packages
- Access Modifiers (default, public, private, protected)



Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"

