

The logo for Oracle Academy is centered on a light gray background. It features the word "ORACLE" in a bold, orange, sans-serif font. Below it, the word "Academy" is written in a smaller, dark gray, sans-serif font. The entire logo is framed by two horizontal dark gray bars, one at the top and one at the bottom.

ORACLE

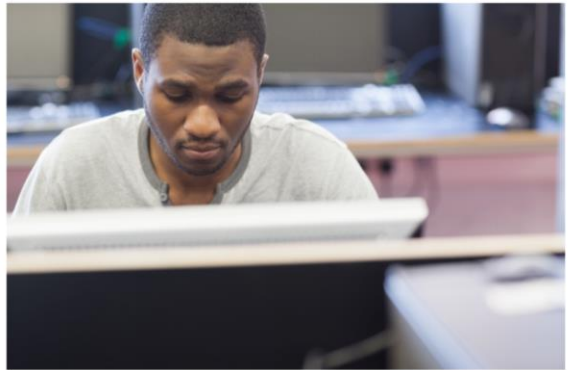
Academy

Java Fundamentals

4-2

Object and Driver Classes

ORACLE
Academy



Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

Objectives

- This lesson covers the following objectives:
 - Describe the general form of a Java program
 - Describe the difference between an Object class and a Driver class
 - Access a minimum of two Java class APIs
 - Explain and give examples of Java keywords
 - Create an Object class
 - Create a Driver class



General Java Program Form

- There are two general forms of Java classes:
 - Driver classes
 - Object classes

Driver Classes

- Driver classes:
 - Contain a main method
 - A main method is necessary to run a Java program in Eclipse
 - The main method may include:
 - Instances of objects from an object class
 - Variables
 - Loops, conditional statements (if-else)
 - Other programming logic
 - Can also contain other static methods

`main()` is similar to `act()` in Greenfoot, as this is where the program begins. However, unlike Greenfoot, the Java program will run only one `main()` method while a Greenfoot program may have many active `act()` methods.

A Java project may have more than one driver class, i.e., one for each object class for testing and debugging, and one for launching the main program.

Object Classes

- Object classes:
 - Are classes that define objects to be used in a driver class
 - Can be found in the Java API, or created by you
 - Examples: String, BankAccount, Student, Rectangle

The Java API is a library of packages and object classes that are already written and are available for use in your programs.



The Java API

- The Java API documentation is located here:

– <http://docs.oracle.com/javase/8/docs/api/>

Packages

Classes

Details for class
selected from the list
of classes

**Java™ Platform, Standard Edition 8
API Specification**

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Profiles

- compact1
- compact2
- compact3

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.

The Java API: String Class

- Scroll through the list of classes until you see String, then click on the link
- You should see the following:

Scroll down in this list to String

Details for the String class

The screenshot shows the Java API documentation for the `String` class. On the left, a list of packages is visible, with `java.lang` selected. The main content area displays the details for the `String` class, including its inheritance hierarchy, implemented interfaces, and source code snippets.

```
java™ Platform
Hotspot 8.0

Overview PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY NESTED | FIELD | CONSTR | METHOD DETAIL FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.lang

Class String
java.lang.Object
java.lang.String

All Implemented Interfaces:
Serializable, CharSequence, Comparable<String>

public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

    String str = "abc";

is equivalent to:

    char data[] = {'a', 'b', 'c'};
    String str = new String(data);

Here are some more examples of how strings can be used:

    System.out.println("abc");
    String cde = "cde";
    System.out.println("abc" + cde);
    String c = "abc".substring(2,3);
    String d = cde.substring(1, 2);
```

ORACLE
Academy

JF 4-2
Object and Driver Classes

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

8

The Java API: Examine String Class

- Examine and discuss the constructors and methods

Scroll down in this list to String

Details for the String class

The screenshot shows the Java Platform Standard Edition 8 API documentation for the `String` class. The left sidebar lists various packages, with `java.lang` selected. The main content area displays the details for the `String` class, including its inheritance hierarchy, implemented interfaces, and source code snippets.

Class String

java.lang.Object
java.lang.String

All Implemented Interfaces:
Serializable, CharSequence, Comparable<String>

public final class **String**
extends Object
implements Serializable, Comparable<String>, CharSequence

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant, their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};  
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");  
String cde = "cde";  
System.out.println("abc" + cde);  
String c = "abc".substring(2,3);  
String d = cde.substring(1, 2);
```

ORACLE
Academy

JF 4-2
Object and Driver Classes

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

9

The Java API: The String Class

- We will use the String class in programs
- The constructor that is most common for this class is:
 - String(String original)
- Common Methods include:

Method	Description
charAt(int index)	Returns the char value at the specified index
length()	Returns the length of this string
substring(int beginIndex)	Returns a new string that is a substring of this string

A Simple Programmer-Created Object Class: Student

- A Java class is used to store or represent data for the object that the class represents
- There are many classes already available from the Java API, but you will want to create many more
- For example, we can create a model, or programmatic representation, of a Student
- Information that we might need for a student includes Student Id, Name, and GPA
- In this lesson, we will create a Object class called Student, then a Driver class called StudentTester

A Simple Programmer-Created Object Class: Student

- All Java classes are created in a text file with the same name as the class
- These files also have a .java extension
- In this lesson, we will create a Student class and a StudentTester class in Eclipse



Eclipse will automatically create and name the .java source code files and, after compilation, the .class byte code files.

Syntax to Create A Simple Programmer-Created Object Class

- The following is example syntax to create a programmer-created object class
- The Java keywords are:
 - package (optional)
 - import (optional)
 - public class

```
package <package_name>;
import <other_packages>;
public class ClassName
{
    <variables (also known as fields)>;
    <constructor method(s)>;
    <other methods>;
}
```

The class variables are sometimes referred to as "instance variables" to distinguish them from "local variables".

Key Terms



Term	Definition
package keyword	<ul style="list-style-type: none">• Defines where this class lives relative to other classes, and provides a level of access control• Use access modifiers (such as public and private) to control access
import keyword	<ul style="list-style-type: none">• Defines other classes or groups of classes that you are using in your class• The import statement provides the compiler information that identifies outside classes used within the current class
class keyword	<ul style="list-style-type: none">• Precedes the name of the class• The name of the class and the file name must match when the class is declared public (which is a good practice)• However, the keyword public in front of the class keyword is a modifier and is not required
class variables or instance fields (often shortened to fields)	<ul style="list-style-type: none">• Variables, or the data associated with programs (such as integers, strings, arrays, and references to other objects)

Packages are covered extensively in Java Programming.

Key Terms



Term	Definition
Constructors	<ul style="list-style-type: none">• Methods called during the creation (instantiation) of an object (a representation in memory of a Java class)
Methods	<ul style="list-style-type: none">• Methods that can be performed on an object• They are also referred to as instance methods• Methods may return an object's variable values (sometimes called functions) or they may change an object's variable values

Java Keywords

- All Java programs use Java keywords
- Examples include the following words: class, public, String, int, for, while, and double
- The font color for Keywords will change in the Eclipse editor
- List of Java keywords:
 - <http://docs.oracle.com/javase/tutorial/java/nutsandbolts/keywords.html>

A Java keyword is a word that has a special function in the Java language, and cannot be used as names for classes, methods, or variables.



ORACLE
Academy

JF 4-2
Object and Driver Classes

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

16

Another name for a keyword is "reserved word".

Java Program Naming Conventions



- Naming Conventions for a Java program:
 - The optional package name is defined before an import statement in lower camel case
 - The optional import statements are defined below the package name
 - The class name is a noun labeled using upper camel case



Camel Case

- Camel case is the practice of stringing capitalized words together with no spaces
- Lower camel case strings capitalized words together, but the lead word is not capitalized
- For example:
 - `thisIsLowerCamelCase`
- Upper camel case strings capitalized words together, but the lead word is capitalized
- For example:
 - `ThisIsUpperCamelCase`

Additional Naming Conventions for a Java Program

- Additional naming conventions for a Java program:
 - Variable names are short but meaningful in lower camel case
 - Constant names are declared in uppercase letters with the final modifier
 - Constructors are named the same as the class name
 - Methods are verbs named in lower camel case



Naming Conventions Example

- The guidelines for a programmer-created object class are labeled in this example for Student
 - Create this file in Eclipse(//is the symbol for comments)

```
package com.example.domain; // Package Declaration
import java.util.Scanner;    // An Import Statement for other packages
public class Student {      // Class Declaration for this file
    private int studentId;   // Variable Declarations for this class
    private String name;
    private String ssn;
    private double gpa;
    public final int SCHCODE = 34958 // A Constant Declaration
    public Student() {       // A Constructor
    } //end constructor
    public int getStudentId() { // An accessor Method
        return studentId;
    } //end method studentId
    public void setStudentId(int x) { // A mutator Method
        studentId = x;
    } //end method setStudentId
} //end class Student
```

Many methods have names that begin with set, get, is, has, etc.

A Simple Programmer-Created Object Class: Student

- The optional package keyword is used in Java to group classes together
 - A package is implemented as a folder
 - Like a folder, it provides a namespace to a class
 - It is recommended to always declare a package at the top of the class

```
package com.example.domain;
```

A Simple Programmer-Created Object Class: Student

- In the example below, a Student class and a Teacher class could be in a folder under the domain name for each developer
- If a company called Acme has developers named Smith and Jones, the packages could be:
 - package com.acme.smith
 - package com.acme.jones
- The path for Jones' file is shown below

Namespace View for Jones:

- com.acme.jones
 - Student
 - Teacher

Folder View for Jones:

```
+com
|_+acme
|_+jones
|_+Student.java
|_+Teacher.java
```

Import Keyword

- The import keyword is used to identify packages or object classes that you want to use in your class
- You can import a single class or an entire package
- You can include multiple import statements
- Import statements follow the package declaration and precede the class declaration

```
import java.util.Scanner;
```

Import Statements

- An import statement is not required, and by default, your class always imports java.lang from the API
 - <http://docs.oracle.com/javase/8/docs/api/java/lang/Package.html>
- You do not need to import classes that are in the same package as the import statement

Import Statement Examples

- Additional examples of import statements:

```
import java.util.Date;    //Import the individual class.
import java.util.*;       //Import the entire package.
import java.util.Date;    //Import using multiple statements.
import java.util.Calendar;
```

While * will import all classes from a package, it is best to name the specific classes in the package to learn them better.

In Eclipse, holding down the ctrl-shift-O (the letter O) will add the appropriate import statements.

Variables for Student Class

- In addition to the package statement and import statements, the Student class will contain variables for student Id, name, social security number, grade point average, and school code
- This will require defining a class with class variables and a constructor
- In addition, methods will be added that can access and change the variables



Do not confuse the slide's use of "class variable" to mean fields with static variables, which are also called class variables.

The Student Class

```
package com.example.domain;
public class Student {
    private int studentId;
    private String name;
    private String ssn;
    private double gpa;
    public final int SCHCODE = 34958;

    public Student() {
    } //end constructor

    public int getStudentId() {
        return studentId;
    } //end method getStudentId

    public void setStudentId(int x) {
        studentId = x;
    } //end method setStudentId
} //end class Student
```

Class Declaration

Fields/Variables

Constructor

Method

Access Modifier

Conventions for the Class Declaration

- Class declaration conventions:
 - The name of the class must be the same as the name of the file that was created in Eclipse
 - The name must begin with a character, and may contain numbers, `_` or `$`
 - Use upper camel case if the name is more than one word
 - For a simple programmer-created object class, the access modifier should be `public` (all access modifiers are either `public`, `private`, or `protected`)

```
public class Student{}
```

All code for this class must be enclosed in a set of curly brackets { }

User named identifiers rarely use `$` or `_` in Java.

Conventions for the Class Variables

- Class variables conventions:
 - Class variables should be declared with the private access modifier to protect the data
 - Class variables are named using lower camel case
 - An exception is a constant (a value that does not change) that should be named using upper case, and declared as public to allow driver programs to access the value

Protecting field values with private is known as encapsulation. This is one of the basic tenets of an object oriented programming language along with inheritance and polymorphism.

Class Variable Declaration Examples

- Examples of declaring class variables:
 - private int length;
 - private int width;
 - private double area;
 - //constant
 - public final double SCALE = 0.25;
 - private String name;



Constructor Methods

- A constructor method is unique in Java because:
 - The method creates an instance of the class
 - Constructors always have the same name as the class and do not declare a return type

```
public Student()  
{  
    //end constructor
```

All code for this method must be enclosed in a set of curly brackets { }

Curly brackets may also be referred to as braces.

Constructor Methods

- With constructor methods:
 - You can declare more than one constructor in a class declaration
 - You do not have to declare a constructor, in fact, Java will provide a default (blank) constructor for you
 - If you declare one or more constructors, Java will not provide a default constructor

```
public Student()  
{  
  
}  
} //end constructor
```

All code for this method must be enclosed in a set of curly brackets { }

Constructors without Parameters

- If you create a constructor without parameters (the parenthesis is empty), you can leave the contents of the constructor (between the { and }) blank
- This is called a default constructor, and is the same as the Java-provided constructor if you do not declare one
- This constructor initializes the numeric class variables to zero, and object variables (such as Strings) to null

```
public Student()  
{  
}  
} //end constructor
```



No Paramters

Constructors without Parameters

- If you create a constructor without parameters (the parenthesis is empty), you can also initialize the variables between the { and }
 - This is also called a default constructor
 - This constructor initializes the numeric class variables to zero, and object variables (such as Strings) to null
 - It has the same results as the previous slide, but the values are more evident

```
public Student(){  
    studentId = 0;  
    name = "";  
    ssn = "";  
    gpa = 0.0;  
} //end constructor
```

No parameters

Constructors with Parameters

- If you create a constructor with parameters (the parenthesis is NOT empty), you would also initialize the variables between the { and }
- This constructor will initialize the class variables with the values that are sent in from the main driver class

```
public Student(int x, String n, String s, double g){  
    studentId = x;  
    name = n;  
    ssn = s;  
    gpa = g;  
} //end constructor
```

Default Constructor Example

- The constructor method in this example is a default constructor that creates an instance of Student

```
package com.example.domain;
public class Student
{
    private int studentId;
    private String name;
    private String ssn;
    private double gpa;
    public final int SCHCODE = 34958;

    public Student() {
    } //end constructor
    public int getStudentId() {
        return studentId;
    } //end method getStudentId
} //end class Student
```

Constructor

Default Constructor Example

- The Student example on the previous slide illustrates a simple no-argument constructor
- The value returned from the constructor is a reference to a Java object of the type created
- Remember, constructors can also take parameters

Constructors

- A constructor is a method that creates an object
- In Java, constructors are methods with the same name as their class used to create an instance of an object
- Constructors are invoked using the new keyword
- Example of code that could be used in a Driver Class to create an object from the Student constructor:

```
Student stu = new Student();
```

Constructors do not have a return type because they always return a new instance of the class (an object).

Main Method

- To run a Java program you must define a main method in a Driver Class
- The main method is automatically called when the class is called
- Remember to name the file the same as the class

You can have Eclipse create the main() method for you when the class is created.

A Simple Programmer-Created Driver Class: StudentTester

- For examples in this course, we will often use the name of the object class followed by the word "Tester."
- Below is an example of a simple Java driver class named StudentTester with a main method

```
public class StudentTester
{
    public static void main(String args[])
    {
        }//end method main
} //end class StudentTester
```


A Simple Programmer-Created Driver Class: StudentTester Example 1

- In this example, a statement is added to create a Student object
- A Student is created, and the class variables are initialized as previously described under default constructors

```
public class StudentTester
{
    public static void main(String args[])
    {
        Student s1 = new Student();
    } //end method main
} //end class StudentTester
```

A Simple Programmer-Created Driver Class: StudentTester Example 2

- In this example the statement to create an object of the Student class is different
- This Student is created using parameters
- Can you guess this student's Id? Name? SSN? GPA?
- Add another student using the default constructor

```
public class StudentTester
{
    public static void main(String args[])
    {
        Student s1 = new Student(123, "Mary Smith", "999-99-9999", 3.4);
    } //end method main
} //end class StudentTester
```

Enhancing the Student Object Class

- Most modern integrated development environments provide an easy way to automatically generate the accessor (getter) and mutator (setter) methods
- Another method that helpful during testing, creating and modifying objects is the toString() method
- In the next few slides, we will develop new methods for the Student object class and modify the StudentTester driver class to test them

Accessor and Mutator Methods

- It is common to create a set of methods that manipulate and retrieve class data values:
 - Accessors (getters):
 - Methods that return (get) the value of each class variable
 - Mutators (setters):
 - Methods that change (set) the value of each class variable

These are needed since we typically have private fields. In addition to simply setting field values, setters could also be used to validate data.

Accessor and Mutator Methods to Enhance the Student Object Class

- Examples (to follow):
 - Add the following Accessor methods:
 - getId()
 - getName()
 - getSSN()
 - getGPA()
 - Add the following Mutator methods:
 - setId()
 - setName()
 - setSSN()
 - setGPA()
 - Add a toString() method to the Student class that will allow us to see the Student data as output

Enhancing the Student Object Class

- Add the `getStudentId()` method and the `setStudentId()` method as shown below
- The object name of the class (`this`) is used to distinguish between the class variable `studentId` and the parameter `studentId` being passed in as an argument

```
public int getStudentId() {  
    return studentId;  
} //end method getStudentId  
  
public void setStudentId(int studentId) {  
    this.studentId = studentId;  
} //end method setStudentId
```

The "this" is necessary when the parameter has the same name as the field. "this" identifies the field.

Additional Methods for the Student Object Class

- Now using the example from the previous slide, add the following methods to the Student object class:
 - getName()
 - getSSN()
 - getGPA()
 - setName()
 - setSSN()
 - setGPA()

Add the toString() Method to the Student Object Class

- Now add the toString() method to the Student object class
- Note, any String object can be built by you to display the information about each student:

```
public String toString()
{
    String s1 = "";
    s1 = "Student Id: " + getId()
        + "Student Name: " + getName()
        + "Student SSN: " + getSSN()
        + "Student GPA: " + getGPA();
    return s1;
} //end method toString
```

The "+" is the String concatenation operator. It joins together two String objects.

Creating the StudentTester Driver Class Main Method

- Now create a StudentTester Driver Class main Method as follows:

```
public class StudentTester
{
    public static void main(String args[])
    {
        Student s1 = new Student(123, "Mary Smith",
                                "999-99-9999", 3.4);
        System.out.println(s1);
        Student s2 = new Student();
        s2.setStudentId(124);
        s2.setStudentName("John Jacoby");
        s2.setSSN("123-45-6789");
        s2.setGPA(4.0);
        System.out.println(s2);
        Student s3 = new Student();
        System.out.println(s3);
    } //end method main
} //end class StudentTester
```

Code Blocks

- A code block is defined by opening and closing "curly braces" { }
- When examining code blocks, consider the following:
 - Every class declaration is enclosed in a code block
 - Method declarations, including the main method, are enclosed in code blocks
 - Java fields and methods have block (or class) scope

```
public class SayHello{                                // Begin class code block
    public static void main(String args[]){           //Begin method code
                                                //block
        System.out.println("Hello Lisa");
    }//end method main                                // End method code block
}//end class StudentTester                            // End class code block
```

Code Block Format

- Code blocks begin with a **{ and end with a }**
- Each time you begin a code block you must have an end
- For example:
 - Every **{ MUST have a matching }**
- Code blocks can be found in:
 - Classes
 - Methods
 - Conditionals (if statements, switch statements)
 - Loops

Terminology

- Key terms used in this lesson included:
 - Access modifiers
 - Code blocks
 - Constants
 - Constructors
 - Driver class
 - import statements
 - Java API
 - Java comments
 - Java keywords
 - Lower camel case
 - Methods

Terminology

- Key terms used in this lesson included:
 - Object class
 - Packages
 - Parameters
 - Programmer-created class
 - Upper camel case
 - Variables

Summary

- In this lesson, you should have learned how to:
 - Describe the general form of a Java program
 - Describe the difference between an Object class and a Driver class
 - Access a minimum of two Java class APIs
 - Explain and give examples of Java keywords
 - Create an Object class
 - Create a Driver class



