MODULE 1: PROGRAMMING IN JAVA

Week 3

Most Essential Learning Competencies: *At the end of the course, you must be able to:*

1. Define abstract class, abstract methods, class, member, attribute, method, constructor, and package
2. Apply the access modifiers private and public as appropriate for the guidelines of encapsulation
3. Use the Java technology application programming interface (API) online documentation

Reading Activity

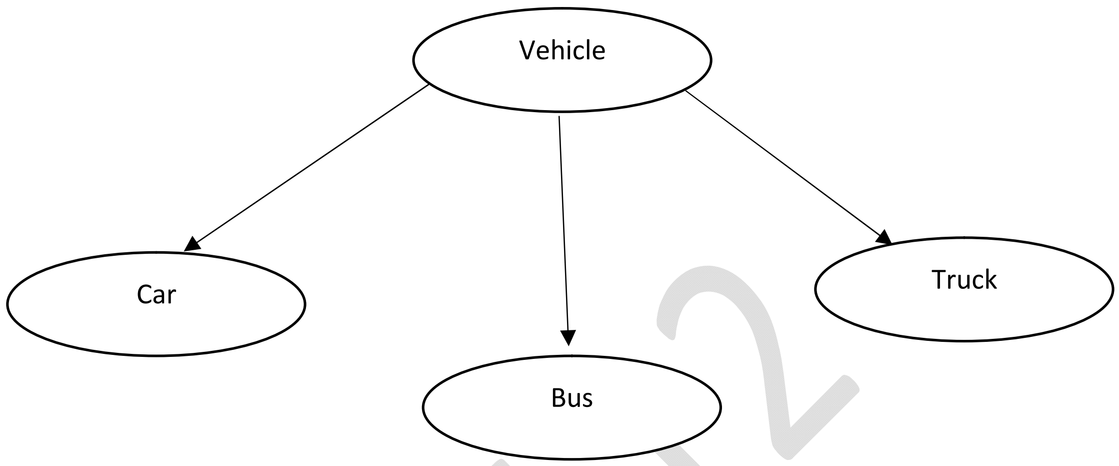


Figure 3.1 Abstract Classes

An abstract class is a class that cannot create new instances. It is used to perform as a base for subclasses. Abstract class cannot be instantiated.

To declare an abstract class, add the abstract keyword to the class declaration.

```
Abstract class SampleAbstractClass {
}
```

Example:

```
abstract class A{
}
class B extends A{
}
```

Abstract Methods

There can be abstract methods in abstract classes. To declare an abstract method, add the abstract keyword before the method declaration.

```
        Abstract class SampleAbstractClass {
                Abstract void abstractMethod();
        }
```

Example:

```
        abstract class Motorcycle{
              abstract void run ();
        }

        class HarleyDavidson extends Motorcycle{
        void run(){System.out.println("Runs fast. .");}

        public static void main(String args[]){
                Motorcycle obj = new HarleyDavidson();
        obj.run();
        }
        }
```

There is no implementation in an abstract method. Methods in a Java interface only have a method signature.

Purpose of Abstract Classes

Abstract classes that operate as base classes can be covered by subclasses to initiate a full implementation. The specific procedure requires three steps: *the step before the action*, *the action*, *and the step after the action*.

If the steps before and after the actions are constantly similar, the three step procedures can be applied in an abstract superclass.

The Analysis and Design Phase

Analysis describes what the system needs to do:

- Modeling the real-world, including actors and activities, objects, and behaviors
- Design describes how the system does it:
  - *Modeling the relationships and interactions between objects and actors in the system*
  - *Finding useful abstractions to help simplify the problem or solution Abstraction*
  - *Functions- Write an algorithm once to be used in many situations*
  - *Objects- Group a related set of attributes and behaviors into a class*

- *Frameworks and APIs- Large groups of objects that support a complex activity*

- *Frameworks can be used as is or be modified to extend the basic behaviour.*

Classes as Blueprints for Objects

In manufacturing, a blueprint describes a device from which many physical devices are constructed. In software, a class is a description of an object:

- *A class describes the data that each object includes.*
- *A class describes the behaviors that each object exhibits.*

In Java technology, classes support three key features of object-oriented programming (OOP):

- *Encapsulation*
- *Inheritance*
- *Polymorphism*

Declaring Java Technology Classes

Basic syntax of a Java class:

```
<modifier>* class <class_name> {
<attribute_declaration>*
<constructor_declaration>*
<method_declaration>*
 }
```

Example:

```
public class Vehicle {
private double maxLoad;
public void setMaxLoad(double value) {
maxLoad = value;
}
}
```

Declaring attributes

Basic syntax of an attribute:

```
<modifier>* <type><name> [ = <initial_value>];
```

Example:

```
public class Foo {
private int x;
private float y = 10000.0F;
private String name = "Bates Motel";
}
```

Declaring Methods

Basic syntax of a method:

```
<modifier>* <return_type><name> ( <argument>* ) {
<statement>*
 }
```

Example:

```
public class Dog {
private int weight;
public int getWeight() {
return weight;
}
public void setWeight(int newWeight) {
if ( newWeight > 0 ) {
weight = newWeight;
}
}
}
```

Accessing Object Members

The dot notation is: <object>.<member>. This is used to access object members, including attributes and methods**.**

Examples of dot notation are:

```
d.setWeight(42);
d.weight = 42; // only permissible if weight is public
```

The problem:

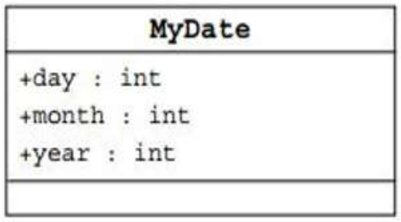


Figure 3.2 Information Hiding

Client code has direct access to internal data (d refers to a MyDate object):

```
d.day = 32;
// invalid day
d.month = 2; d.day = 30;
```

```
// plausible but wrong
d.day = d.day + 1;
// no check for wrap around
```

The solution:

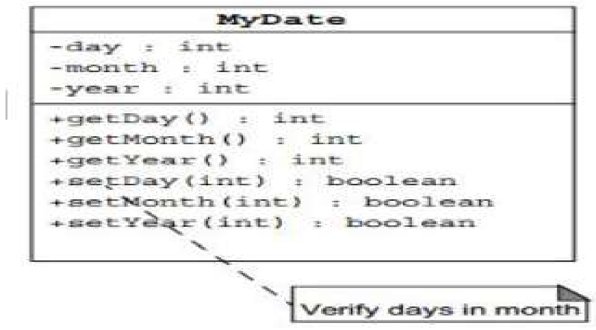


Figure 3.3 Information Hiding

Client code must use setters and getters to access internal data:

```
MyDate d = new MyDate();
d.setDay(32);
// invalid day, returns false
d.setMonth(2);
d.setDay(30);
// plausible but wrong,
// setDay returns false
d.setDay(d.getDay() + 1);
// this will return false if wrap around
// needs to occur
```

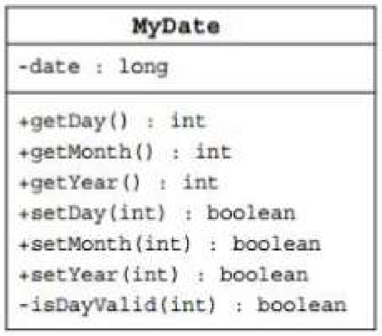


Figure 3.4 Encapsulation

– *Hides the implementation detail of a class*

- *Forces the user to use an interface to access data*
- *Makes the code more maintainable*

Declaring Constructors

Basic syntax of a constructor:

```
[<modifier>] <class_name> ( <argument>* ) {
<statement>*
}
```

Example:

```
public class Dog{
private int weight;
public Dog() {
weight = 42;
}
}
```

The Default Constructor

- *There is always at least one constructor in every class.*
- *If the writer does not supply any constructors, the default constructor is present automatically:*
- *The default constructor takes no arguments*
- *The default constructor body is empty*
- *The default enables you to create object instances with new Xxx() without having to write a constructor*

Source File Layout

Basic syntax of a Java source file list is:

```
[<package_declaration>]
<import_declaration>*
<class_declaration>+
```

For example, the VehicleCapacityReport.java file is:

```
package shipping.reports;
import shipping.domain.*;
import java.util.List;
import java.io.*;
public class VehicleCapacityReport {
private List vehicles;
```

public void generateReport(Writer output) {...}
}

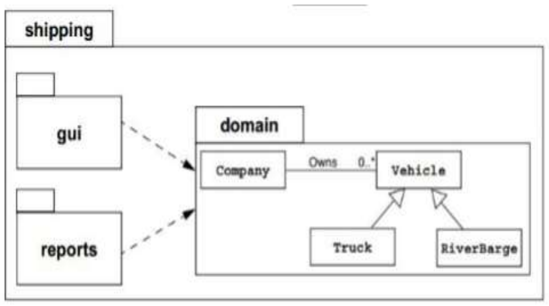


Figure 3.5 Software Packages

- *Packages help manage large software systems.*
- *Packages can contain classes and sub-packages.*

The Package Statement
Basic syntax of the package statements is:
package<top_pkg_name>[.sub_pkg_name>]*;

Examples of the statement are:

- *Package shipping.gui.reportscreens;*
- *Specify the package declaration at the beginning of the source file.*
- *Only one package declaration per source file.*
- *If no package is declared, then the class is placed into the default package.*
- *Package names must be hierarchical and separated by dots.*

The import Statement

Basic syntax of the import statement is:
import <pkg_name>[.<sub_pkg_name>]*.<class_name>; OR
import <pkg_name>[.<sub_pkg_name>]*.*;

Examples of the statement are:
import java.util.List;
import java.io.*;
import shipping.gui.reportscreens.*;

The import statement does the following:

- *Precedes all class declarations*
- *Tells the compiler where to find classes*

Directory Layout Packages
Packages are stored in the directory tree containing the package name.

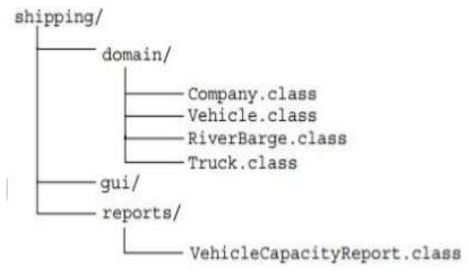


Figure 3.6 Shipping Application Packages

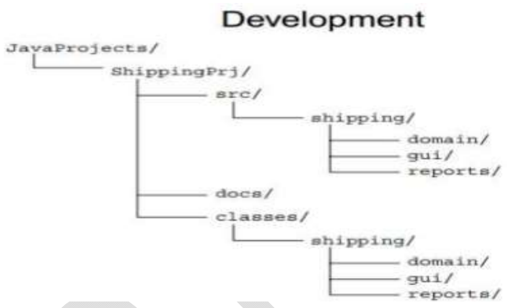


Figure 3.7 Compiling Using the -d Option

```
cd JavaProjects/ShippingPrj/src
javac -d ../classes shipping/domain/*.java
```

Terminology Recap

1. ***Class*** *– The source-code blueprint for a run-time object*
2. ***Object*** *– An instance of a class; also known as instance*
3. ***Attribute*** *– A data element of an object; also known as data member, instance variable, and data field*
4. ***Method*** *– A behavioral element of an object; also known as algorithm, function, and procedure*
5. ***Constructor*** *– A method-like construct used to initialize a new object*
6. ***Package*** *– A grouping of classes and sub-package*
7. ***Instantiate*** *– to create an instance by of an object.*

Using the Java Technology API Documentation

1. A set of Hypertext Markup Language (HTML) files provides information about the API.
2. A frame describes a package and contains hyperlinks to information describing each class in that package.
3. A class document includes the class hierarchy, a description of the class, a list of member variables, a list of constructors, and so on

Watch Video Resources

1. **Classes and Objects**

   https://youtu.be/Hhja1xMjJF0

2. **Building Reusable Methods in Java**

   https://youtu.be/xqlznn85EzI

Self-Check

Quiz 3.1

Instructions: *Write your answer on the Answer Sheet (AS) provided in this module.*

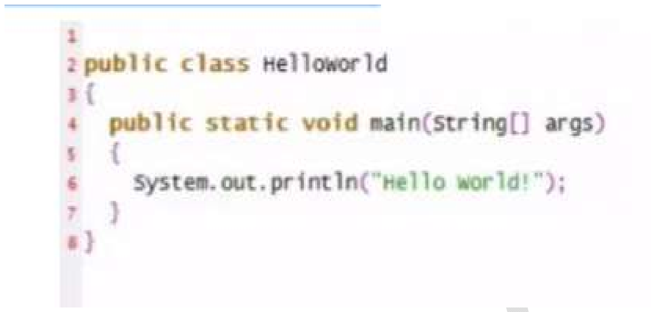
Fill in the blanks with the correct answers. (2-points each)

1. Abstract classes that operate as ______ can be covered by subclasses to initiate a full implementation.
2. Classes support three key features of object-oriented programming (OOP) _______, ________, _________.
3. Encapsulation _______ the implementation detail of a class.
4. Instantiate to create an ______ by of an object.
5. Package a grouping of classes and _______.
6. Class the source-code ______for a run-time object.
7. _______ a method-like construct used to initialize a new object.
8. Method a _________of an object.
9. Object an _________also known as instance
10. Attribute a ________ of an object also known as data member, instance variable, and data field.

Laboratory Activity

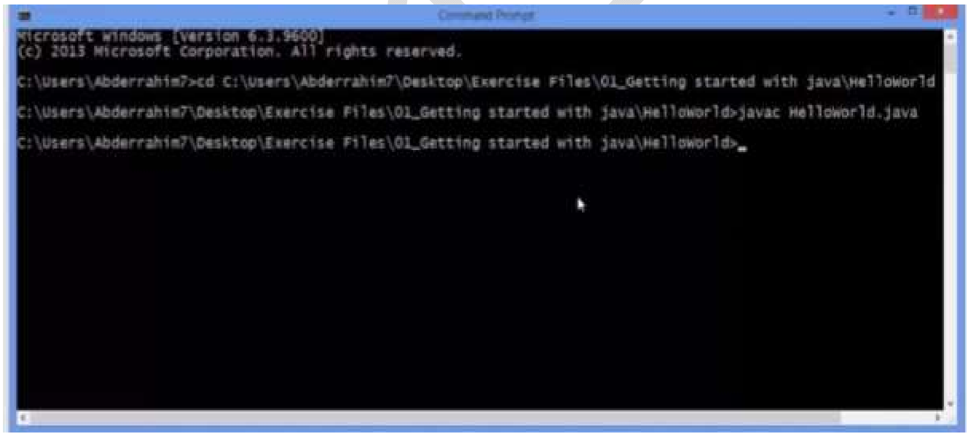
Activity 3.1

1. Open available IDE and type the following codes:



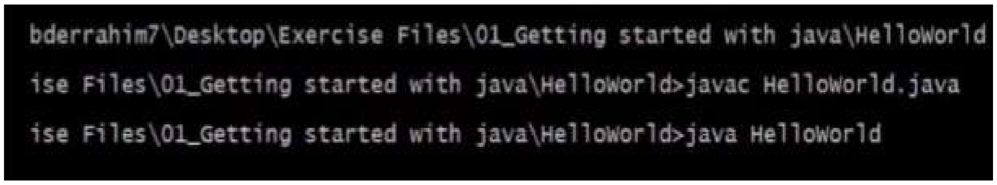
```
public class Helloworld
{
  public static void main(String[] args)
  {
    System.out.println("Hello world!");
  }
}
```

2. Save file as **Helloworld.java** to a destination folder
3. Open Command Prompt. Change directory to where **Helloworld.java** folder is located. Then, compile by typing **javac Helloworld.java**

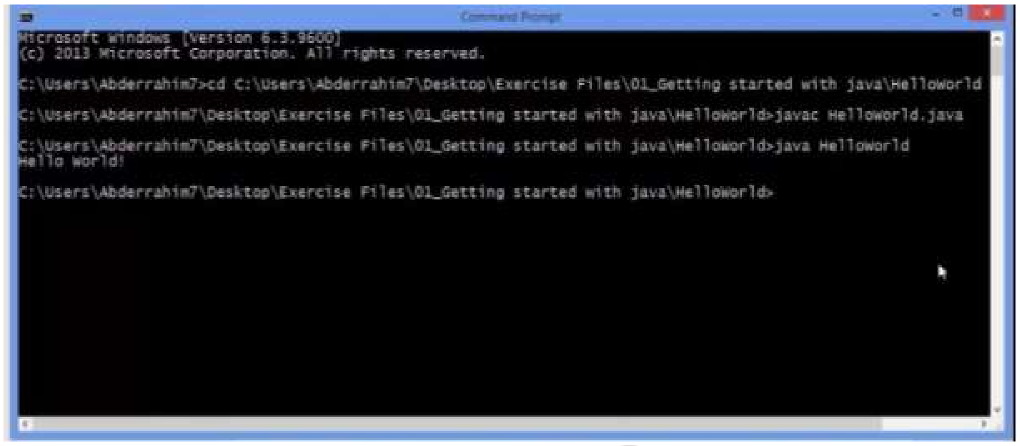


```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Abderrahim7>cd C:\Users\Abderrahim7\Desktop\Exercise Files\01_Getting started with java\HelloWorld

C:\Users\Abderrahim7\Desktop\Exercise Files\01_Getting started with java\HelloWorld>javac HelloWorld.java

C:\Users\Abderrahim7\Desktop\Exercise Files\01_Getting started with java\HelloWorld>_
```

4. To display the output, type **java Helloworld**



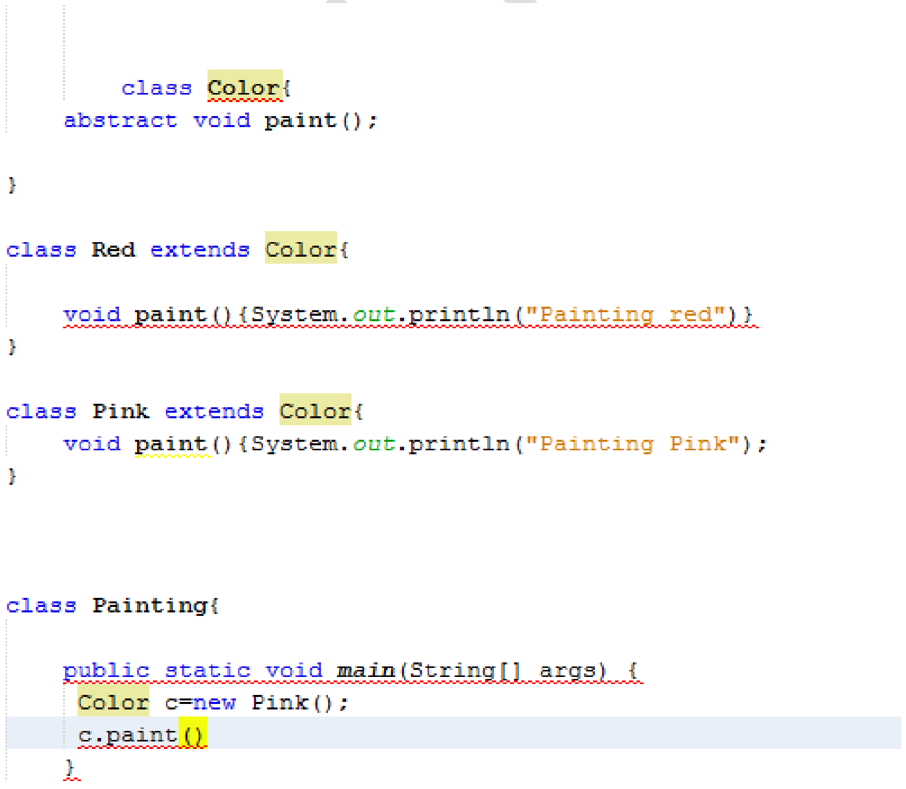
```
bderrahim7\Desktop\Exercise Files\01_Getting started with java\HelloWorld

ise Files\01_Getting started with java\HelloWorld>javac HelloWorld.java

ise Files\01_Getting started with java\HelloWorld>java HelloWorld
```



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Abderrahim7>cd C:\Users\Abderrahim7\Desktop\Exercise Files\01_Getting started with java\HelloWorld

C:\Users\Abderrahim7\Desktop\Exercise Files\01_Getting started with java\HelloWorld>javac HelloWorld.java

C:\Users\Abderrahim7\Desktop\Exercise Files\01_Getting started with java\HelloWorld>java HelloWorld
Hello World!

C:\Users\Abderrahim7\Desktop\Exercise Files\01_Getting started with java\HelloWorld>
```

Activity 3.2

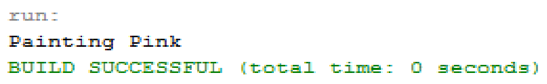
1. Open available IDE and type the following codes:



```
    class Color{
abstract void paint();

}

class Red extends Color{

    void paint(){System.out.println("Painting red")}
}

class Pink extends Color{
    void paint(){System.out.println("Painting Pink");
}



class Painting{

    public static void main(String[] args) {
     Color c=new Pink();
     c.paint()
    }
```

2. Save the file as **Painting.java**

3. If there are errors, debug the program. Compile it again. Do this repeatedly until no errors will be detected.
4. It should display the expected output:



```
run:
Painting Pink
BUILD SUCCESSFUL (total time: 0 seconds)
```

Internet References

1. https://www.youtube.com
2. https://www.youtube.com/c/devfactor/about
3. Phoenix Publishing House Inc.