# Object Oriented Programming using Java

**Prepared By:**
**Suyel, PhD**
**Assistant Professor**
**Dept. of CSE, NIT Patna**

# Outline

1. Source File Structure
2. Basic Terminology
3. Simple Java Program

# Source File Structure

- The entire program may consist of more than one source file.
- Each source file must have the same name as a public class that it declares.
- Source file can contain only one public class declaration.
- The file extension must be ".java".
- The filename is case-sensitive.
- There are mainly three major sections in a java source file:
  1. Package declaration
  2. Import declaration
  3. Top level class and interface declaration

# Source File Structure (Cont…)

## Package Declaration

- Java source file should start with a package declaration.
- A package in java is an encapsulation mechanism that can be used to group related classes, interfaces and subpackages.
- If the package declaration is default, package declaration is optional.

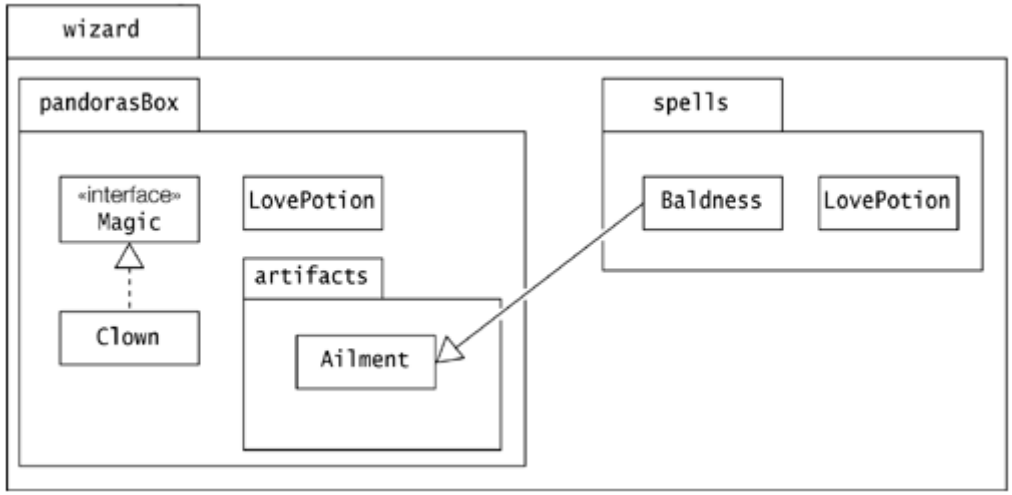


**Fig. 1.** Package hierarchy

# Source File Structure (Cont…)

## Package Declaration (Cont…)

- Example: package wizard;
- Fig. 1 shows an example of a package hierarchy, comprising of a package called wizard.
- It contains two other packages: pandorasBox and spells.
- The package pandorasBox has a class called Clown that implements an interface called Magic, also found in the same package.
- The package pandorasBox has a class called LovePotion and a subpackage called artifacts containing a class called Ailment.
- The package spells has two classes: Baldness and LovePotion. The class Baldness is a subclass of class Ailment found in the subpackage artifacts in the package pandorasBox.

# Source File Structure (Cont…)

## Package Declaration (Cont…)

- Package in java can be categorized into two forms:

1. **Built-in package:** There are many built-in packages, such as java, lang, awt, javax, swing, net, io, util, sql, etc.

2. **User-defined package:** We can create our own package.

# Source File Structure (Cont…)

## Import Declaration

- After the packages are declared, there are import declarations.
- We use the import statement to tell the compiler where to find the external classes required by the source program under compilation.
- Import declarations can be zero or more.
- These import statements introduce type or static member names in the source code.
- It is mandatory to place all import declarations before actual class declarations in java source code.
- To import a single class, we specify the name of the class.
- To import all classes, we specify "*".

# Source File Structure (Cont…)

## Import Declaration (Cont…)

- Examples:

```
import mypackage.MyClass;
import mypackage.reports.accounts.salary.EmployeeClass;
import java.io.BufferedWriter;
import java.awt.*;
```

- In C, "#include" loads all the specified header files at the time of include statement only irrespective of whether we are using these header files or not. Hence, this is static binding.
- In java, "import statement" loads no ".class" file at the time of import statement. In the next lines of the code, when we use a class at that time only corresponding ".class" files are loaded. This type of loading is called dynamic loading or load on demand or load on fly.

# Source File Structure (Cont…)

## Import Declaration (Cont…)

- There are two types of import declarations:

1. **Explicit class import:** This type of import is highly recommended to use as it improves readability of the code.

   Example: import java.util.ArrayList;

2. **Implicit class import:** It is never recommended to use this type of import because it reduces readability of the code.

   Example: import java.util.*;

# Source File Structure (Cont…)

## Top Level Class and Interface Declaration

- The declarations of class, interface and enum, etc. are collectively known as type declarations.
- Since these declarations belong to the same package, they are said to be defined at the top level, which is the package level.
- There can be any number of such type declarations.
- The order of type declarations is not mandatory.
- Technically, a source file need not have any such definitions, but that is hardly useful.

# Source File Structure (Cont…)

- Example:

```
//Part 1 (Optional)
package wizard;
//Part 2 (Zero or More)
import java.util.*;
//Part 3 (Zero or More)
public class Hello
{
 }
```

# Basic Terminology

## Class

- A class denotes a category of objects, and acts as a blueprint for creating such objects.
- A class models an abstraction by defining the properties and behaviors for the objects representing the abstraction.
- An object exhibits the properties and behaviors defined by its class.
- The properties of an object of a class are also called attributes, and are defined by **fields** in java.
- A field in a class definition is a variable, which can store a value that represents a particular property.
- The behaviors of an object of a class are also known as operations, and are defined using **methods** in java.
- Fields and methods in a class definition are collectively called members.

# Basic Terminology (Cont…)

## Class (Cont…)

- The class definition also has a declaration like method with the same name as the class, which is known as a **constructor**.
- A constructor is executed, when an object is created from the class. At the time of calling constructor, memory for the object is allocated.

```
//Source Filename: CharStack.java
public class CharStack                  //Class name and class declaration
{
    //Fields:
    private char[] stackArray;      // The array implementing the stack
    private int    topOfStack;      // The top of the stack

    //Constructor:
    public CharStack(int n)         { stackArray = new char[n]; topOfStack = -1; }

    //Methods:
    public void push(char element) { stackArray[++topOfStack] = element; }
    public char pop()               { return stackArray[topOfStack--]; }
    public char peek()              { return stackArray[topOfStack]; }
    public boolean isEmpty()        { return topOfStack < 0; }
    public boolean isFull()         { return topOfStack == stackArray.length - 1; }
}
```

# Basic Terminology (Cont…)

## Object

- An object is an instance of a class and the process of creating objects from a class is called **instantiation**.
- The object is constructed using the class as a blueprint and is a concrete instance of the abstraction that the class represents.
- An object must be created before it can be used in a program.
- In java, objects are manipulated through object references (also called reference values or simply references).
- Each object has a unique identity and has its own copy of the fields declared in the class definition.
- The purpose of the constructor call on the right side of the **new** operator is to initialize the newly created object.

# Basic Terminology (Cont…)

## Object (Cont…)

- The process of creating objects usually involves the following steps:
  1. **Declaration of a variable:** This involves declaring a reference variable of the appropriate class to store the reference to the object.

     Declaration of two reference variables denotes two distinct objects.

     Example: CharStack stack1, stack2;
  2. **Creating an object:** Here, we use the **new** operator in conjunction with a call to a constructor to create an instance of the class.

     Example: stack1 = new CharStack(10);

     The **new** operator returns a reference to a new instance of the CharStack class. This reference can be assigned to a reference variable of the appropriate class.

     The declaration and the instantiation can also be combined.

     Example: CharStack stack1 = new CharStack(10);

# Basic Terminology (Cont…)

- **Instance Members**
  - Each created object has its own copy of the fields defined in its class.
  - The fields of an object are called **instance variables**.
  - The values of the instance variables in an object comprise its state. Two distinct objects can have the same state, if their instance variables have the same values.
  - The methods of an object define its behavior. These methods are called **instance methods**.
  - Instance variables and instance methods, which belong to objects are collectively called instance members.

# Basic Terminology (Cont…)

## Static Member

- In some cases, certain variables should only belong to the class and not be part of any object created from the class, which called **static variables**.
- It belongs to the class, and not to any object of the class.
- A static variable is initialized, when the class is loaded at runtime.
- Similarly, a class can have **static methods** that belong only to the class, and not to any objects of the class.
- Static variables and static methods are collectively known as **static members**, and are distinguished from instance members in a class definition by the keyword **static** in their declaration.

# Basic Terminology (Cont…)

## Static Member (Cont…)

- An object can be made to exhibit a particular behavior by invoking the appropriate operation on the object. In java, this is done by calling a method on the object using the binary infix dot **"."** operator.

```
CharStack stack = new CharStack(5);      //Create a stack
stack.push('J');                         //Character 'J' pushed
char c = stack.pop();                    //One character popped and returned: 'J'
stack.printStackElements();              //Compile time error: No such method in CharStack
```

- The 1st method call pushes one character on the stack and it does not return anything.
- The 2nd method call pops one character off the stack and returns the character popped.
- Trying to invoke a method printStackElements() on the stack results in a compile-time error, as no such method is defined in the class.

# Simple Java Program

- To create a simple java program, we need to create a class that contains main method.

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

# Simple Java Program (Cont…)

- **Save:** Hello.java
- **To compile:** javac Hello.java
- **To execute:** java Hello
- **Output:** Hello World

# Simple Java Program (Cont…)

- **class** keyword is used to declare a class.
- **public** keyword is an access modifier, which represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The main advantage of the static method is that there is no need to create an object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create an object to invoke the main method. Thus, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String args []** is an array of strings, which stores arguments passed by command line while starting a program.
- **System.out.println()** is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class.

# Simple Java Program (Cont…)

- The subscript notation can be used after type, before the variable or after the variable.
  - public static void main(String[] args)
  - public static void main(String []args)
  - public static void main(String args[])

# Simple Java Program (Cont…)

- Valid main method signature:
  - public static void main(String[] args)
  - public static void main(String []args)
  - public static void main(String args[])
  - public static void main(String... args)
  - static public void main(String[] args)
  - public static final void main(String[] args)
  - final public static void main(String[] args)
  - final strictfp public static void main(String[] args)

# Simple Java Program (Cont…)

- Invalid main method signature:
  - public void main(String[] args)
  - static void main(String[] args)
  - public void static main(String[] args)
  - abstract public static void main(String[] args)

# Simple Java Program (Cont…)

- Giving a semicolon at the end of a class is optional in java.

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

thank you . . .

**Slides are prepared from various sources, such as Book, Internet Links and many more.**