

Java Programming Language



Instructor: C. Pu (Ph.D., Assistant Professor)

puc@marshall.edu



First Java Program

- Let us look at a simple code that will print ***Hello World***.

```
public class MyFirstJavaProgram {  
    /*  
        This is the first java program.  
        This will print 'Hello World' as the output  
    */  
    public static void main(String []args) {  
        System.out.println("Hello World"); // prints Hello World  
    }  
}
```



Basic Syntax

- About Java programs, it is very important to keep in mind the following points.
 - **Case Sensitivity** - Java is case sensitive, which means identifier *Hello* and *hello* would have different meaning in Java.
 - **Class Names** - For all class names, the first letter should be in *Upper Case* letter.
 - If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.
 - **Method Names** - All method names should start with a *Lower Case* letter.
 - If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.



Basic Syntax

- About Java programs, it is very important to keep in mind the following points.
 - **Program File Name** - Name of the program file should *exactly match* the class name.
 - Example:
 - Assume '**MyFirstJavaProgram**' is the class name. Then the file should be saved as '**MyFirstJavaProgram.java**'
 - **public static void main(String args[])** - Java program processing starts from the **main()** method which is a *mandatory* part of every Java program.



Identifiers

- All Java components require names.
- Names used for classes, variables, and methods are called **identifiers**.



Identifiers

- In Java, there are several points to remember about **identifiers**:
 - All identifiers should begin with a *letter* (A to Z or a to z), *dollar sign* (\$) or an *underscore* (_).
 - After the first character, identifiers can have any combination of characters.
 - A **keyword** cannot be used as an identifier.
 - Identifiers are **case sensitive**.
 - Examples of legal identifiers:
 - age, \$salary, _value, __l_value.
 - Examples of **illegal** identifiers:
 - 123abc, -salary.



Modifiers

- It is possible to modify classes, methods, etc., by using modifiers.
- There are two categories of modifiers:
 - **Access Modifiers:**
 - default, public, protected, private
 - **Non-access Modifiers:**
 - static, final, abstract



Variables

- Following are the types of variables in Java:
 - **Local variables**
 - A variable defined within a block or method or constructor is called local variable.
 - **Class variables (Static variables)**
 - A static variable is declared using the **static** keyword within a class outside any method, constructor or block.
 - **Instance variables (Non-static variables)**
 - Instance variables are non-static variables and are declared in a class outside any method, constructor or block.



Arrays

- Arrays are objects that store a fixed-size sequential collection of elements of the same type.
- Declaring array variables

```
dataType[] arrayRefVar; // preferred way.  
or  
dataType arrayRefVar[]; // works but not preferred way.
```

- Example:

```
double[] myList; // preferred way.  
or  
double myList[]; // works but not preferred way.
```

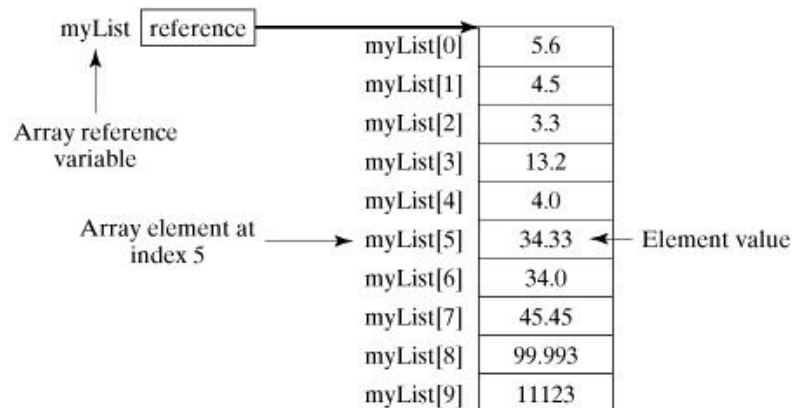
Arrays (cont.)

- Creating arrays
 - You can create an array by using the **new** operator

```
arrayRefVar = new dataType[arraySize];
```

- Example:

```
double[] myList = new double[10];
```





Arrays (cont.)

- When processing array elements, we often use **for** loop

```
public class TestArray {  
    public static void main(String[] args) {  
        double[] myList = {1.9, 2.9, 3.4, 3.5};  
        // Print all the array elements  
        for (int i = 0; i < myList.length; i++) {  
            System.out.println(myList[i] + " ");  
        }  
        // Summing all elements  
        double total = 0;  
        for (int i = 0; i < myList.length; i++) {  
            total += myList[i];  
        }  
        System.out.println("Total is " + total);  
    }  
}
```



Keywords

- The following list shows the reserved words in Java.
 - These reserved words may not be used as constant or variable or any other identifier names.

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while



Comments

- Java supports **single-line** and **multi-line** comments.
- All characters available inside any comment are **ignored** by Java compiler.

```
public class MyFirstJavaProgram{  
    /* This is my first java program.  
    * This will print 'Hello World' as the output  
    * This is an example of multi-line comments.  
    */  
    public static void main(String []args){  
        // This is an example of single line comment  
        /* This is also an example of single line comment. */  
        System.out.println("Hello World");  
    }  
}
```



Using Blank Lines

- A line containing only white space, possibly with a comment, is known as a blank line, and Java totally ignores it.
- ***Read `JavaCodeConventions.pdf`***



Objects in Java

- Object
 - Objects have states and behaviors.
 - Example:
 - A dog has **states** - color, name, breed as well as **behaviors** – wagging the tail, barking, eating.
- If you compare the software object with a real-world object, they have very similar characteristics.
 - Software objects also have a **state** and a **behavior**.
 - A software object's state is stored in **fields** and behavior is shown via **methods**.



Classes in Java

- Class
 - A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.
- A class is a blueprint from which individual objects are created.



Classes in Java

- Following is a sample of a class.

```
public class Dog{  
    String breed;  
    int age;  
    String color;  
    void barking(){  
    }  
    void hungry(){  
    }  
    void sleeping(){  
    }  
}
```



Classes in Java (cont.)

- A class can contain any of the following variable types:
 - **Local variables:**
 - Variables defined inside methods, constructors or blocks are called local variables.
 - The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
 - **Instance variables:**
 - Instance variables are variables within a class but outside any method.
 - These variables are initialized when the class is instantiated.
 - Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
 - **Class variables:**
 - Class variables are variables declared within a class, outside any method, with the **static** keyword.



Constructors

- Every class has a **constructor**.
- If we do not explicitly write a constructor for a class, the Java compiler builds a **default constructor** for that class.
- Each time a new object is created, at least one constructor will be invoked.
- The main rule of constructors is that they should have the **same name** as the class.
- A class can have more than one constructor.



Constructors (cont.)

- Following is an example of a constructor:

```
public class Puppy{  
  
    public Puppy()  
    {  
  
    public Puppy(String name){  
        // This constructor has one parameter, name.  
    }  
  
}
```



Creating an Object

- A class provides the blueprints for objects.
- So basically, an object is created from a class.
- In Java, the **new** keyword is used to create new objects.
 - **Declaration:** A variable declaration with a variable name and an object type.
 - **Instantiation:** The **'new'** keyword is used to create the object.
 - **Initialization:** The **'new'** keyword is followed by a call to a constructor.
 - This call initializes the new object.



Creating an Object (cont.)

- Following is an example of creating an object:

```
public class Puppy{  
    public Puppy(String name){  
        // This constructor has one parameter, name.  
        System.out.println("Passed Name is :" + name );  
    }  
  
    public static void main(String []args){  
        // Following statement would create an object myPuppy  
        Puppy myPuppy = new Puppy("tommy");  
    }  
}
```



Accessing Instance Variables and Methods

- Instance variables and methods are accessed via created objects.

```
/* First create an object */
```

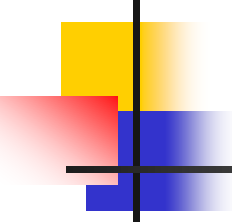
```
ObjectReference = new Constructor();
```

```
/* Now call a variable as follows */
```

```
ObjectReference.variableName;
```

```
/* Now you can call a class method as follows */
```

```
ObjectReference.MethodName();
```



Accessing Instance Variables and Methods (cont.)

- This example (Puppy.java) explains how to access instance variables and methods of a class.



Source File Declaration Rules

- Let's now look into the source file declaration rules.
 - There can be only one public class per source file.
 - A source file can have multiple non-public classes.
 - The public class name should be the name of the source file as well which should be appended by .java at the end.
 - If the class is defined inside a package, then the package statement should be the first statement in the source file.
 - If import statements are present, then they must be written between the package statement and the class declaration. If there are no package statements, then the import statement should be the first line in the source file.
 - Import and package statements will imply to all the classes present in the source file. It is not possible to declare different import and/or package statements to different classes in the source file.



Import Statements

- In Java, if a fully qualified name, which includes the package and the class name, is given, then the compiler can easily locate the source code or classes.
- Import statement is a way of giving the proper location for the compiler to find that particular class.
- For example, the following line would ask the compiler to load all the classes available in directory `java_installation/java/io`:

```
import java.io.*;
```



Basic Datatypes

- Variables are nothing but reserved memory locations to store values.
 - This means that when you create a variable you reserve some space in the memory.
- Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.
 - By assigning different datatypes to variables, you can store integers, decimals, or characters in these variables.
- There are two data types available in Java:
 - Primitive Datatypes
 - Reference/Object Datatypes



Primitive Datatypes

- There are eight primitive datatypes supported by Java.
- Primitive datatypes are predefined by the language and named by a keyword.
 - byte
 - short
 - int
 - long
 - float
 - double
 - boolean
 - char



Reference Datatypes

- Reference variables are created using defined constructors of the classes.
 - They are used to access objects.
 - These variables are declared to be of a specific type that cannot be changed.
- Class objects and various type of array variables come under reference datatype.
- Default value of any reference variable is null.
- A reference variable can be used to refer any object of the declared type or any compatible type.
 - Example: `Animal animal = new Animal("giraffe");`



Java Literals

- A literal is a source code representation of a fixed value.
- They are represented directly in the code without any computation.
- Literals can be assigned to any primitive type variable.
- For example:

```
int i = 68;  
char a = 'A'
```