**SCSBC2 PRELIMS**

---

## Intro to Programming

***Program***
- Set of step-by-step instruction that directs the computer to do the task you want it to do and to produce the results wanted.

***Programming Language***
- Set of rules that provides a way of telling the computer what to perform.
- Set of commands that is formulated by the user and executed by the computer.

***Programming***
- The task of creating software.

## Why Programming?
- It helps us understand more the computer which is considered as a computing tool.
- Let's you find out quickly whether you like programming or not and whether you have analytical turn of mind the programmers need.

## What Programmers Do?
- makes / executes programs
- converts problems and solution to computer instruction
- prepares instruction of program, runs, tests and debug the program
- write a report on the program
- perform changes on programs as a result of new requirements

## Programming Process

***Defining the Problem***
- Determine the requirement the program must meet
  - Input – data/requirements that are inputted
  - Process – the operations
  - Output – shows the result

***Planning the solution***
- Select the best method for solving the problem
  - Algorithm/ Pseudocode
  - Flowcharting

***Coding the program***
- Prepare the set of instructions for the computer to execute

***Requirements in coding***
- Correct use of language
- Coded program must be keyed in a form that the computer can understand

***Testing the program***
- Perform debugging and testing the program, using representative input data
- Methods:
  - **Disk Checking** – proofreading
  - **Translating** – translate a coded program to be understood by the computer, thru the use of a translator
    - Two types of Translator
      - **Interpreters** – translate the program one at a time
      - **Compilers** – translate the entire program at one time
  - **Debugging** – directs, locates and corrects bugs by running the program
    - **Bugs** – errors in programs
    - 2 types of Bugs:
      - **Syntax Errors** – errors in spelling or grammar
      - **Semantic Errors** – logical error, incorrect solution

**Signs of Semantically Wrong Statements**
- Incorrect results are displayed
- Confusing or garbled output
- Program hangs
- Run time errors are displayed and program terminates

***Documenting the program***
- Write up the full specifications for other program users
- Used to make programs readable by other programmers
  - **Documentation** – description of the programming cycle and specifies accts about the program.
  - **Comment** – an important part of the program documentation

## Java
- Is a high-level programming language originally developed by Sun Microsystems and released in 1995 as core component of Sun Microsystems' Java platform
- Is a **programming language** and **platform**.
  - any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.
- Latest release of the Java Standard Edition is Java 19.
- Runs on a variety of platforms, such as Windows, Mac OS and the various versions of UNIX

***James Gosling***
- Canadian computer scientist, founder and lead designer of Java programming language.

## Java is
***Object oriented***

***Platform independent*** – when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code.
  - Byte code is distributed over the web and interpreted by the Virtual Machine

(JVM) on whichever platform it is being run on.

***Simple*** (easy to learn)

***Secure*** – it enables to develop virus-free, tamper-free systems

***Architecture-neutral*** – generates object file format, which makes the compiled code executable

***Portable***

***Robust*** – makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

***Multithreaded*** – multithreaded feature it is possible to write programs that can perform many tasks simultaneously

***Interpreted*** – Java byte code is translated on the fly to native machine instructions and is not stored anywhere.

***High performance*** – with the use of Just-In-Time compilers, Java enables high performance.

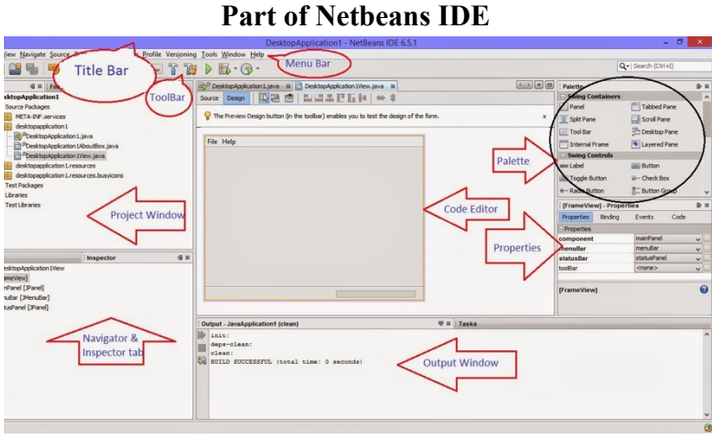
***Distributed*** – Java is designed for the distributed environment of the internet.

***Dynamic*** – Java is considered to be more dynamic and designed to adapt to an evolving environment

## History of Java

- The language, initially called 'Oak' after an oak tree that stood outside Gosling's office, also went by the name 'Green' and ended up later being renamed as Java, from a list of random words.
- Sun released the first public implementation as Java 1.0 in 1995.
- It promised **Write Once, Run Anywhere (WORA)**, providing no-cost run-times on popular platforms.
- On November 13, 2006, Sun released much of Java as free and open source software under the terms of the GNU **General Public License (GPL).**

## Part of Netbeans IDE




## Program Sample

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

## Basic Syntax

- Java program is defined as a collection of objects that communicate via invoking each other's methods.

***Objects***
- Objects have states and behaviors. Example: A dog has states – color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.

***Class***
- A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.

***Methods***
- A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

***Instance Variables***
- Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

## First Java Program

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

## Factors to be Considered

***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. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

**Example**: class MyFirstJavaClass

***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 word's first letter should be in Upper Case.

**Example**: public void myMethodName()

***Program File Name***
- Name of the program file should exactly match the class name.
  - When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).

**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.

## Java Identifiers

***Identifiers***
- Names used for classes, variables, and methods

***Rules in Naming Identifiers***
- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- A key word cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.

## Java Modifiers

**Two Categories of Modifiers**

***Access Modifiers***
- default, public, protected, private
- in Java the accessibility or scope of a field, method, constructor, or class.
- We can change the access level of fields, constructors, methods, and class by applying the access by applying the access modifier on it.

***Non-access Modifiers***
- final, abstract, strictfp
- such as static, abstract, synchronized, native, volatile, transient, etc.

## Four Types of Java Access Modifiers

***Private***
- The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

**Example:**

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}
public class Simple{
     public static void main(String args[]){
       A obj=new A();
       System.out.println(obj.data); //Compile Time Error
       obj.msg();//Compile Time Error
     }
}
```

*In the example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.*

***Default***
- The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- Is accessible only within package
- Cannot be accessed from outside the package
- Provides more accessibility that private
- Is more restrictive that protected, and public

**Example:**

```
//save by A.java
package pack;
class A{
      public static void main(String args[]){
        System.out.println("Hello");
      }
}
//save by B.java
package mypack;
import pack.*;
class B{
      public static void main(String args[]){
        A obj = new A();//Compile Time Error
       obj.msg();//Compile Time Error
      }
}
```

*In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package. In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.*

***Protected***
- The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- Is accessible within package and outside the package but through inheritance only
- Protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.
- Provides more accessibility than the default modifier.

**Example:**

```
//save by A.java
package pack;
public class A{
  protected void msg() {
          System.out.println("Hello");
  }
}

//save by B.java
package mypack;
import pack.*;
class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

*In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.*

***Public***
- The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.
- Is accessible everywhere.
- Has the widest scope among all other modifiers.

**Example:**

```
//save by A.java
package pack;
public class A{
  public void msg() {
     System.out.println("Hello");
  }
}

//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

## Java Literals

***Java Variables***
- Variables are containers for storing data values.

**Different types of variables**
***String***
- stores text, such as "Hello". String values are surrounded by double quotes

***int***
- stores integers (whole numbers), without decimals, such as 123 or -123

***float***
- stores floating point numbers, with decimals, such as 19.99 or -19.99.

***char***
- stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes.

***boolean***
- stores values with two states: true or false

### Declaring (Creating) Variables

- To create a variable, you must specify the type and assign it a value:

**Syntax** :

```
type variableName = value;
```

- Where type is one of Java's types (such as int or String), and variableName is the name of the variable (such as x or name). The equal sign is used to assign values to the variable.

### Final Variables

- If you don't want others (or yourself) to overwrite existing values, use the final keyword (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

**Example:**

```
public class Main {
  public static void main(String[] args) {
    final int myNum = 15;
    myNum = 20; // will generate an error
    System.out.println(myNum);
  }
}
```

### Display Variables

- The println() method is often used to display variables.
- To combine both text and a variable, use the + character:

### Display Variables for numeric values

- For numeric values, the + character works as a mathematical operator.

## Declare Many Variables

- To declare more than one variable of the same type, you can use a comma-separated list.

**Example:**
int x = 5, y = 6, z = 50;

**Example:**
int x, y, z;
x = y = z = 50;

## Java Data Types

***Primitive Data Types***

- A primitive data type specifies the size and type of variable values, and it has no additional methods.

| Data Type | Size | Description |
|---|---|---|
| byte | 1 byte | Stores whole numbers from -128 to 127 |
| short | 2 bytes | Stores whole numbers from -32,768 to 32,767 |
| int | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |
| long | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits |
| Boolean | 1 bytes | Stores true or false values |
| char | 2 bytes | Stores a single character/letter or ASCII values |

***Non-Primitive Data Types***

- Non-primitive data types are called reference types because they refer to objects.
- The main difference between primitive and non-primitive data types are:
  - Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
  - Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
  - A primitive type has always a value, while non-primitive types can be null.
  - A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
  - The size of a primitive type depends on the data type, while non-primitive types have all the same size.
- Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc.

## Java User Input (Scanner)

- The Scanner class is used to get user input, and it is found in the java.util package.
- To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings:

**Example:**

```
import java.util.Scanner;  // Import the Scanner class

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");
    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName); // Output user input
  }
}
```

**Input Types**

| Method | Description |
|---|---|
| nextBoolean() | Reads a boolean value from the user |
| nextByte() | Reads a byte value from the user |
| nextDouble() | Reads a double value from the user |
| nextFloat() | Reads a float value from the user |
| nextInt() | Reads a int value from the user |
| nextLine() | Reads a String value from the user |
| nextLong() | Reads a long value from the user |
| nextShort() | Reads a short value from the user |

## Java Operators

- Operators are used to perform operations on variables and values.

***Arithmetic Operators***

- Arithmetic operators are used to perform common mathematical operations.

| Operator | Name | Description | Example |
|---|---|---|---|
| + | Addition | Adds together two values | x + y |
| - | Subtraction | Subtracts one value from another | x – y |
| * | Multiplication | Multiplies two values | x * y |
| / | Division | Divides one value by another | x / y |
| % | Modulus | Returns the division remainder | x % y |
| ++ | Increment | Increases the value of a variable by 1 | ++x |
| -- | Decrement | Decreases the value of a variable by 1 | --x |

***Assignment Operators***

- Assignment operators are used to assign values to variables.

| Operator | Example | Same as |
|---|---|---|
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x – 3 |
| *= | x *= 3 | x = x * 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| &= | x &= 3 | x = x & 3 |
| \|= | x \|= 3 | x = x \| 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

***Comparison Operators***

- Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.
- The return value of a comparison is either true or false. These values are known as Boolean values, and you will learn more about them in the Booleans and If..Else chapter.

| Operator | Name | Example |
|---|---|---|
| == | Equal to | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

***Logical Operators***

- You can also test for true or false values with logical operators.
- Logical operators are used to determine the logic between variables or values

| Operator | Name | Description | Example |
|---|---|---|---|
| && | Logical and | Returns true if both statements are true | x < 5 && x < 10 |
| \|\| | Logical or | Returns true if one of the statements is true | x < 5 \|\| x < 4 |
| ! | Logical not | Reverse the result, returns false if the result is true | ! (x < 5 && x < 10) |

***Special Characters***

| Escape Character | Result | Description |
|---|---|---|
| \’ | ‘ | Single quote |
| \” | “ | Double quote |
| \\ | \ | Backslash |

| Code | Result |
|---|---|
| \n | New Line |
| \r | Carriage Return |
| \t | Tab |
| \b | Backspace |

## Java Method

***Method***

- Is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Used to perform certain actions.
- Also known as **functions**.

***Why use methods?***

- To reuse code: define the code once, and use it many times.

***Call a Method***

- To call a method in Java, write the method’s name followed by two parenthesis () and a semicolon ;
- In the following example, myMethod() is used to print a text, when it is called.

## Java Method Parameters

***Parameters and Arguments***

- Information can be passed to methods as parameter.
- Parameters act as variables inside the method.
- Parameters are specified after the method name, inside the parentheses.
- You can add as many parameters as you want, just separate them with a comma.

```
public class Main {
 static void myMethod(String fname) {
   System.out.println(fname + “Refsnes”);
 }
 public static void main(String[] args) {
    myMethod(“Liam”);
    myMethod(“Jenny”);
    myMethod(“Anja”);
 }
}
```

***Multiple Parameters***

```
public class Main {
 static void myMethod(String fname, int age) {
   System.out.println(fname + “is” + age);
 }
 public static void main(String[] args) {
    myMethod(“Liam”, 5);
    myMethod(“Jenny”, 8);
    myMethod(“Anja”, 31);
 }
}
```

***Return Values***

- The void keyword, used in the examples above, indicates that the method should not return a value.
- If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method.

```
public class Main {
 static int myMethod(int x) {
    return 5 + x;
  }
  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }
}
```

***Java String toLowerCase() Method***

- The toLowerCase() method converts a string to lower case letters

```
public class Main {
  public static void main(String[] args) {
    String txt = “Hello World”;

    System.out.println(txt.toLowerCase());
  }
}
```

***Java String toLowerCase() Method***

- The toUpperCase() method converts a string to upper case letters

```
public class Main {
  public static void main(String[] args) {
    String txt = “Hello World”;

    System.out.println(txt.toUpperCase());
  }
}
```

**Java Math**

- The Java Math class has many methods that allows you to perform mathematical task on numbers.

***Math.max(x,y)***

- The Math.max(x,y) method can be used to find the highest value of x and y.

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.max(5, 10));
  }
}
```

***Math.min(x,y)***

- The Math.min(x,y) method can be used to find the lowest value of x and y.

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.min(5, 10));
  }
}
```

***Math.sqrt(x)***

- The Math.sqrt(x) method returns the square root of x.

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.sqrt(64));
  }
}
```

***Math.pow(x,y)***

- The Math.pow(x,y) method returns the value of the power of y.

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.pow(5, 2));
  }
}
```

***Math.abs(x)***

- The Math.abs(x) method returns the absolute (positive) value of x.

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.abs(-4, 7));
  }
}
```

***Math.random(x,y)***

- The Math.random() method returns a random number between 0.0 (inclusive), and 1.0 (exclusive).

```
public class Main {
  public static void main(String[] args) {
    System.out.println(Math.random());
  }
}
```

**Good luck, CS 1-2!**