

**GROW  
CONFI  
DENTLY**

# Module Objective

At the end of this module, you will be able to:

- - Describe the primary concepts that support Java technology
- - Explain how Java achieves platform independence
- - Discuss the different tools and libraries available as part of the Standard Java Development KIT (J2SE JDK)
- - Install and configure the required software, tools, and libraries to get started with Java



→ Write, compile, and execute simple Java applications

# Module Objectives (cont.)

- At the end of this module, participants will be able to:
  - Explain the concept of controlling ‘program flow’
  - Control ‘program flow’ by using the different control flow statements
  - Describe the concept of methods
  - Identify and create correctly structured methods
  - Describe the concept of ‘method calling’



# Java Source File Structure

- Java applications are composed of text files ending with a '.java' suffix that contain source codes
- Each Java source file consists of a 'class declaration', which follows a specific structure
- Java source files are compiled into '.class' files which are then run by the Java interpreter



Refer to the MainSample.java sample code

# Java Source File Structure

## declaration order

### 1. Package declaration

Used to organize a collection of related classes.

### 2. Import statement

Used to reference classes and declared in other packages.

### 3. Class declaration

A Java source file can have several classes but only one public class is allowed.

```
/*
 * Created on Jun 25, 2008
 *
 * First Java Program
 */
package sef.module3.sample;
import java.lang.*;

/**
 * @author SEF
 */
public class MainSample{

    public static void main(String[] args) {
        // print a message
        System.out.println("Welcome to Java!");
    }

}
```



# Java Source File Structure

## Comments

### 1. Block Comment

```
/*  
 * insert comments  
here  
*/
```

### 2. Documentation Comment

```
/**  
 * insert  
documentation  
*/
```

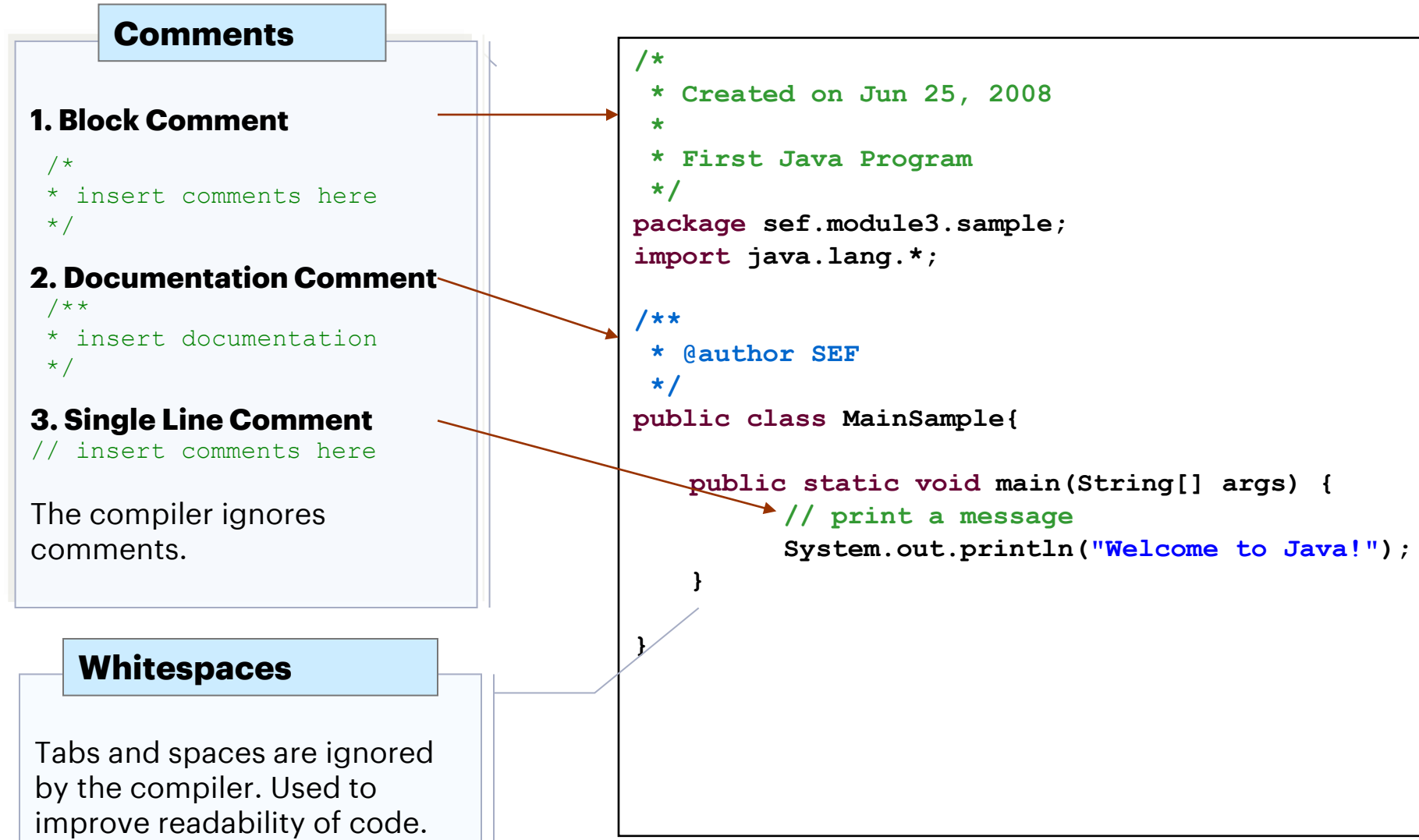
### 3. Single Line Comment

```
// insert comments  
here
```

The compiler ignores  
comments.

```
/*  
 * Created on Jun 25, 2008  
 *  
 * First Java Program  
 */  
package sef.module3.sample;  
import java.lang.*;  
  
/**  
 * @author SEF  
 */  
public class MainSample{  
  
    public static void main(String[] args) {  
        // print a message  
        System.out.println("Welcome to Java!");  
    }  
}
```

# Java Source File Structure



# Java Source File Structure

## Class

- Class is the fundamental component of all Java programs.
- Every Java program includes at least one public class definition.
- A class definition contains all the variables and methods that make the program work. This is contained in the class body indicated by the opening and closing braces.
- The name of the public class declaration should be the same as the name of the file (case sensitive).

```
/*
 * Created on Jun 25, 2008
 *
 * First Java Program
 */
package sef.module3.sample;
import java.lang.*;

/**
 * @author SEF
 */
public class MainSample{

    public static void main(String[] args){
        // print a message
        System.out.println("Welcome to Java!");
    }

}
```



# Java Source File Structure

## Braces

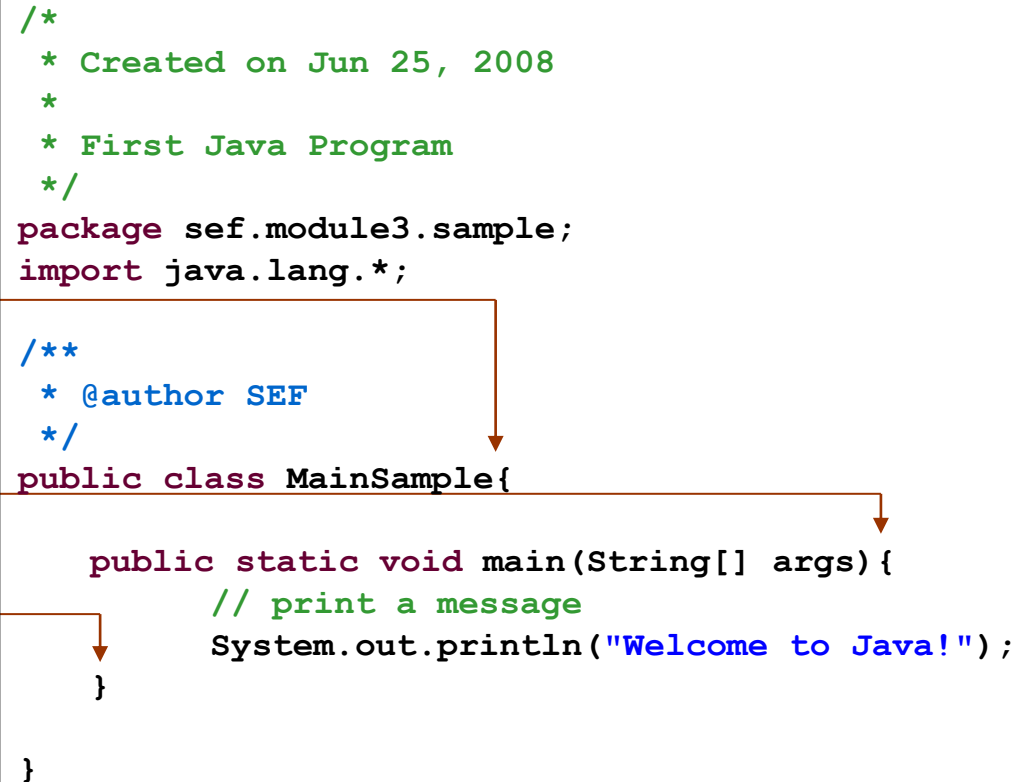
- Braces are used for grouping statements or block of codes.
- The left brace ( { ) indicates the beginning of a class body, which contains variables and methods of the class.
- The left brace also indicates the beginning of a method body.
- For every left brace that opens a class or method you need a corresponding right brace ( } ) to close the class or method.
- A right brace always closes its nearest left brace.

```
/*
 * Created on Jun 25, 2008
 *
 * First Java Program
 */
package sef.module3.sample;
import java.lang.*;

/**
 * @author SEF
 */
public class MainSample{

    public static void main(String[] args){
        // print a message
        System.out.println("Welcome to Java!");
    }

}
```



# The 'main()' Method

- The 'main' method is where the execution of a java application begins

```
Syntax: public static void main( String[] args ) {  
    //Main method implementation goes here  
}
```

- Classes that have a main method declared inside serve as the starting point of the application



Refer to the MainSample.java sample code.

# The 'main()' Method

## main() method

This line begins the `main()` method. This is the line at which the program will begin executing.

## String args[]

Declares a parameter named `args`, which is an array of `String`. It represents command-line arguments.

```
/*
 * Created on Jun 25, 2008
 *
 * Hello World Program
 */
package sef.module3.sample;

/**
 * @author SEF
 */
public class MainSample {

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

## Terminating character

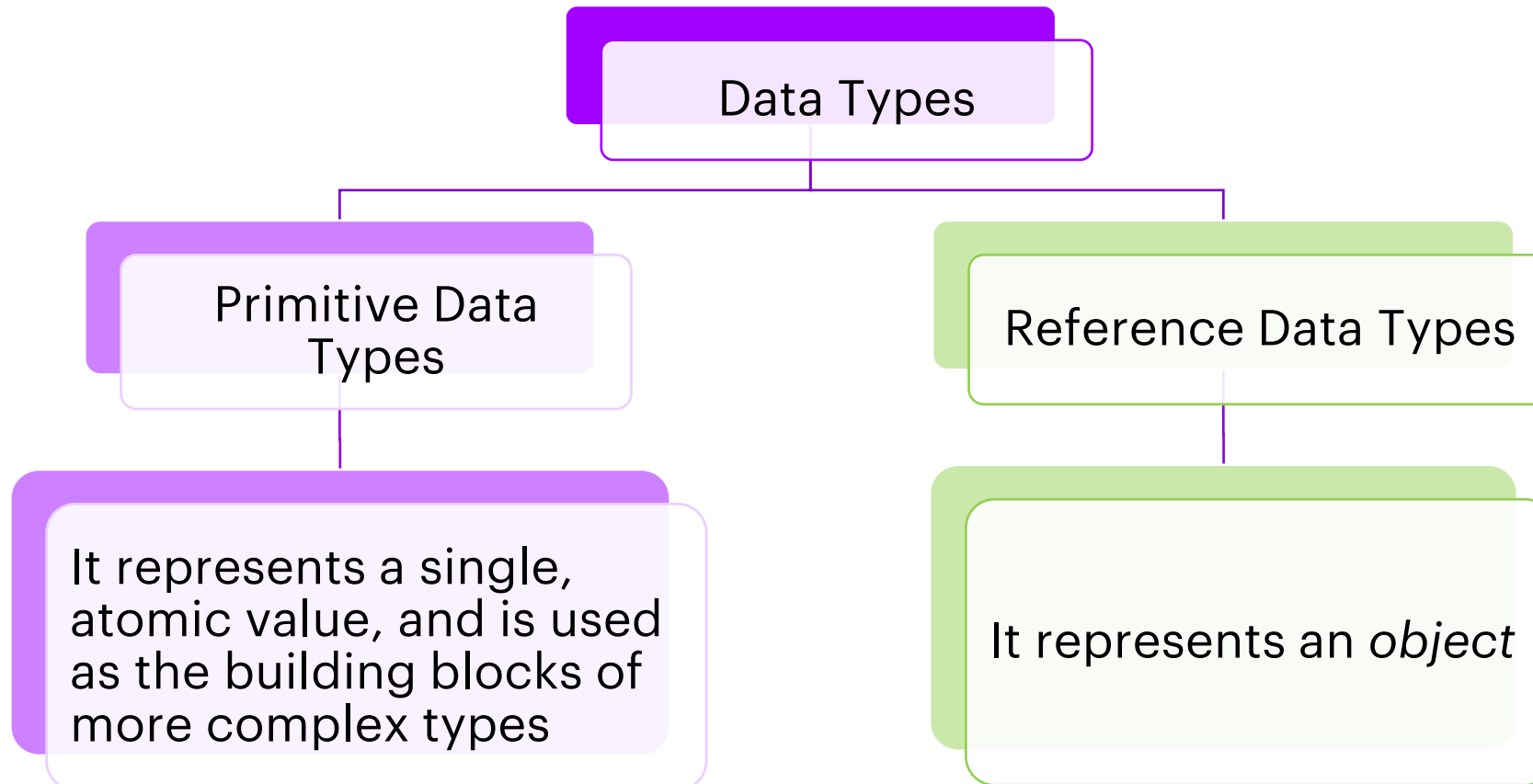
Semicolon (;) is the terminating character for any java statement.

# Java Keywords

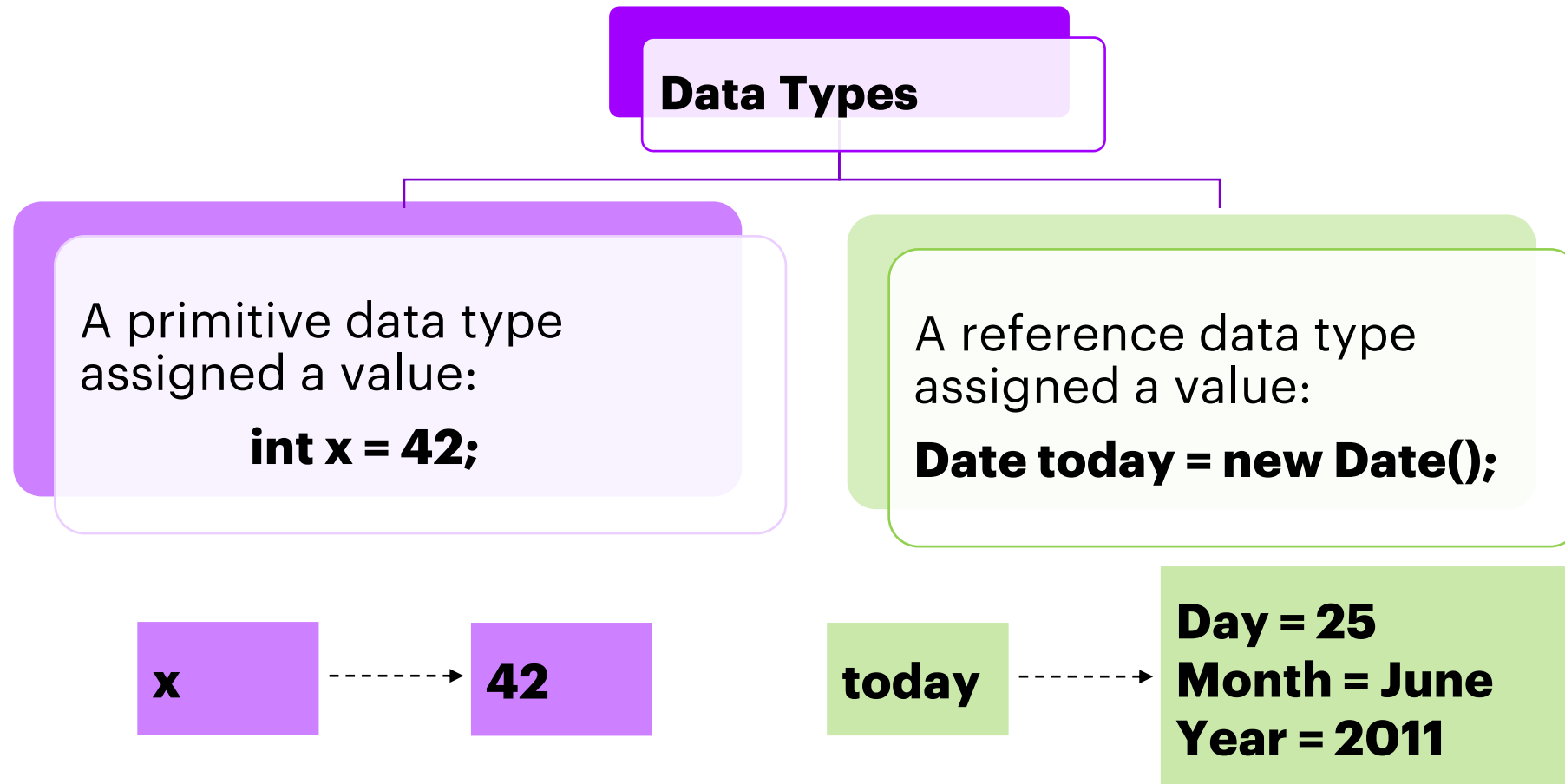
<code>abstract</code>	<code>default</code>	<code>if</code>	<code>package</code>	<code>synchronized</code>
<code>assert</code>	<code>do</code>	<code>implements</code>	<code>private</code>	<code>this</code>
<code>boolean</code>	<code>double</code>	<code>import</code>	<code>protected</code>	<code>throw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

# Data Types

- A data type determines the values that a variable can contain, and the operations that can be performed on it.



# Data Types





# Primitive Data Types

Type	Bits	Lowest Value	Highest Value
<b>boolean</b>	(n/a)	<b>false</b>	<b>true</b>
<b>char</b>	16	'\u0000' [0]	'\uffff' [ $2^{16}-1$ ]
<b>byte</b>	8	-128 [ $-2^7$ ]	+127 [ $2^7-1$ ]
<b>short</b>	16	-32,768 [ $-2^{15}$ ]	+32,767 [ $2^{15}-1$ ]
<b>int</b>	32	-2,147,483,648 [ $-2^{31}$ ]	+2,147,483,647 [ $2^{31}-1$ ]
<b>long</b>	64	-9,223,372,036,854,775,808 [ $-2^{63}$ ]	+9,223,372,036,854,775,807 [ $2^{63}-1$ ]
<b>float</b>	32	$\pm 1.40129846432481707e-45$	$\pm 3.40282346638528860e+38$
<b>double</b>	64	$\pm 4.94065645841246544e-324$	$\pm 1.79769313486231570e+308$

# Variables

- A named storage location used to represent data that can be changed while the program is running
- Declaration specifies a variable's properties, like its data type and the name with which it would be identified
- Basic variable declaration in Java:

**Syntax:**                    **<data type> <identifier\_name>;**

**Examples:**                **int myInteger;**  
                              **String myFirstName;**  
                              **Date theDateToday;**

# Variables: Initialization

## Initializing variables with primitive data type

**Syntax:**           <identifier\_name> = <initial\_value>;  
**Example:**       myInteger = 0;

## Initializing variables with reference data type

**Syntax:**           <identifier\_name> = <initial\_value>;  
**Examples:**       myFirstName = "Jason";  
                      theDateToday = new Date( );

# Variable Assignment

- Variables are assigned a value using the assignment operator equal ( = )

**Syntax:**            **<variable\_name> = <the\_value>;**  
**Example:**        **myInteger = 0;**

- The data type of the value being assigned must be compatible with the data type of the variable receiving the value

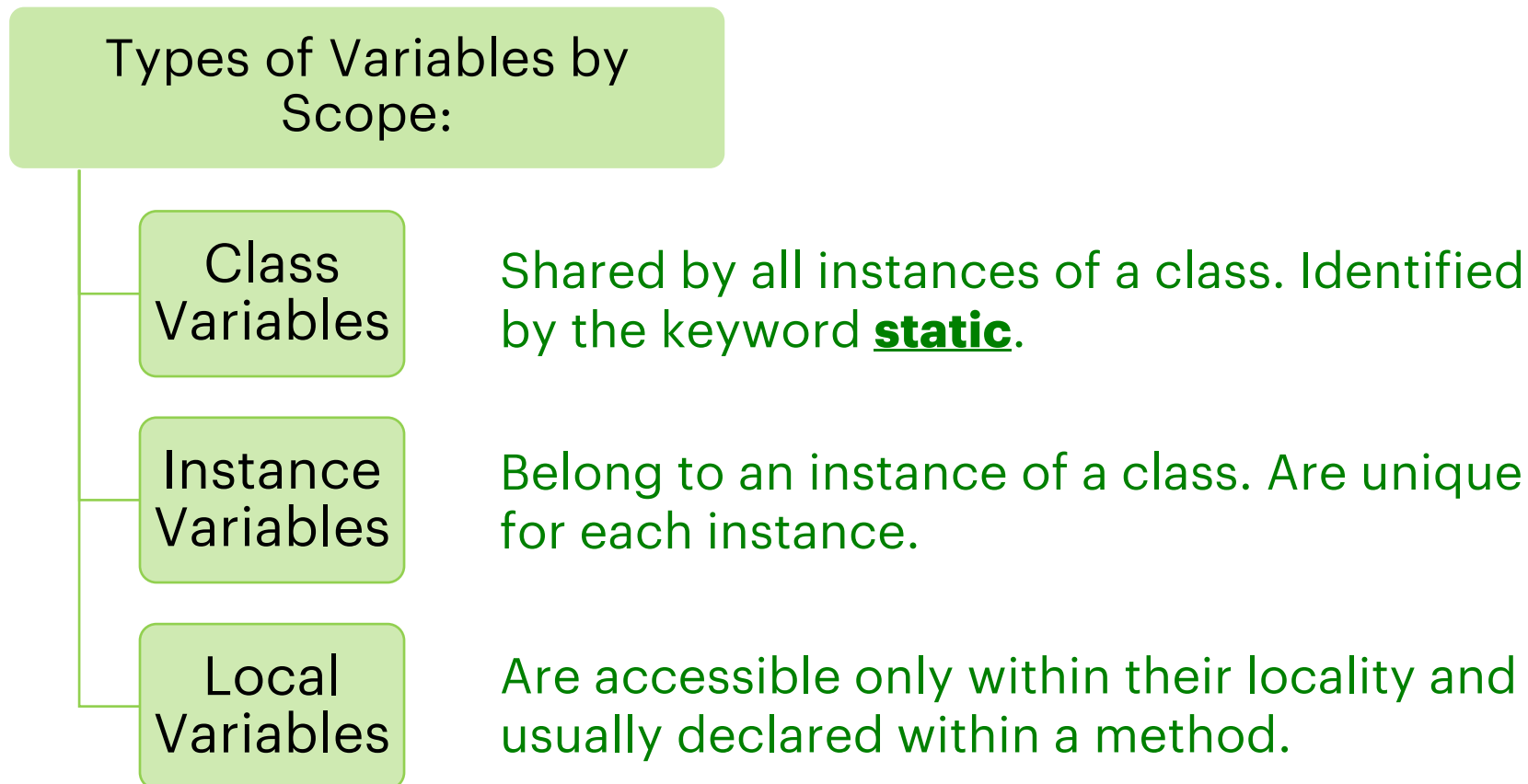
# Activity 1

- Open the file 'VariableAssignmentActivity.java' in the package sef.module3.activity
- Perform the following:
  - Declare a variable of type int and assign it default value
  - Update the value
  - Print the updated value to the console



# Variables: Scope

- Refers to portions or sections of a program where the variable has value and is said to be 'visible'





# Variables: Scope

## Class Variable

Declared outside methods but inside a class. Denoted by **static** keyword.

## Local Variable

Declared inside methods and/or subroutines. *aString* is local to the method *main*

## Instance Variables

Declared outside methods but inside a class.

```
package sef.module3.sample;
```

```
public class VariableScope {  
    /**  
     * @param args  
     */
```

```
    public static void main(String[] args) {  
        String aString = "This is a local variable";  
    }
```

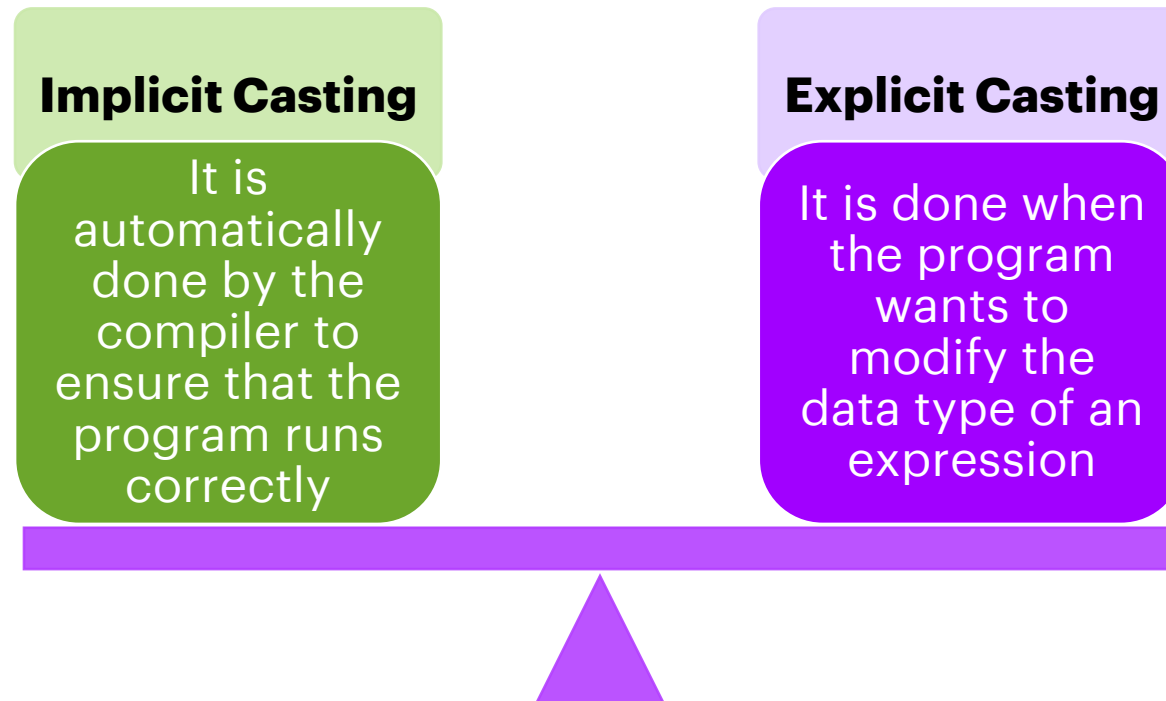
```
}  
  
class Employee {  
    public static int totalCount = 0;  
    private String myFirstName;  
    private String myLastName;  
    private int myAge;  
}
```

# Expressions and Statements

- An expression is a 'value' whereas a statement is an 'action'
- A Statement is a complete unit of instruction, usually ended by a semicolon ( ; )
- The example below is a statement assigning the value of the expression to the variable
  - *String greeting = "hello";*
  - *final static double PIE = 3.14*
  - *z = x + y;*
  - *result = getFactorial(z);*
  - *Date today = new Date();*
- Method calls are also statements
  - *System.out.println("Hello");*

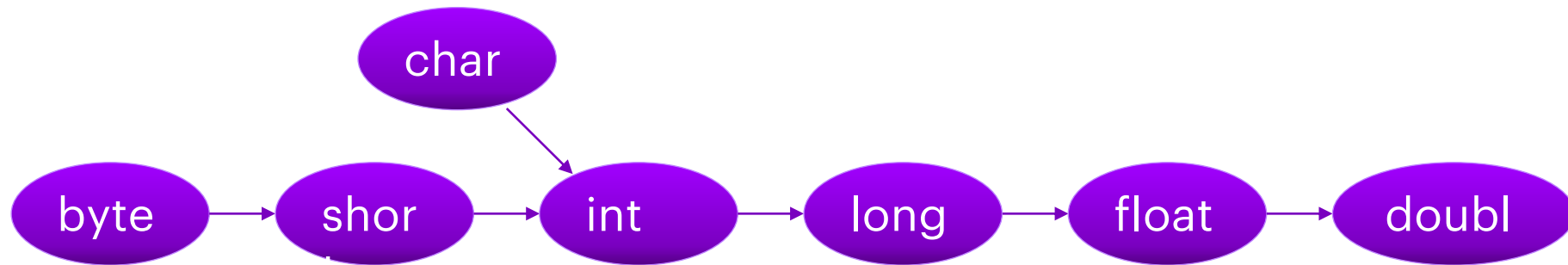
# Type Casting

- Refers to the conversion of the data type of a **variable** value to another data type.
- Java is strongly typed, meaning it expects that values being assigned to each other be compatible. Casting is a way to express this compatibility.



# Primitive Data Type Casting Flow

*widening conversion*



*narrowing conversion*

# Implicit Casting

- Implicit casting is an implied casting operation.
- Implicit casting usually takes place when casting from a narrower data type to a broader data type, this can also be referred to as widening conversion.

<b>Example A:</b>	<b>int a = 1;</b> <b>double b = a;</b>
<b>Example B:</b>	<b>float x = 1.0;</b> <b>double y = x;</b>

- Since a double is 'wider' than either an integer or a floating point, it is able to accept and cast their values implicitly.

# Explicit Casting

- Explicit casting is a required casting operation.
- It usually takes place when casting from a broader data type to a narrower data type, this can also be referred to as narrowing conversion.

**Syntax:** <destination variable> = (<destination data type>) <source variable>

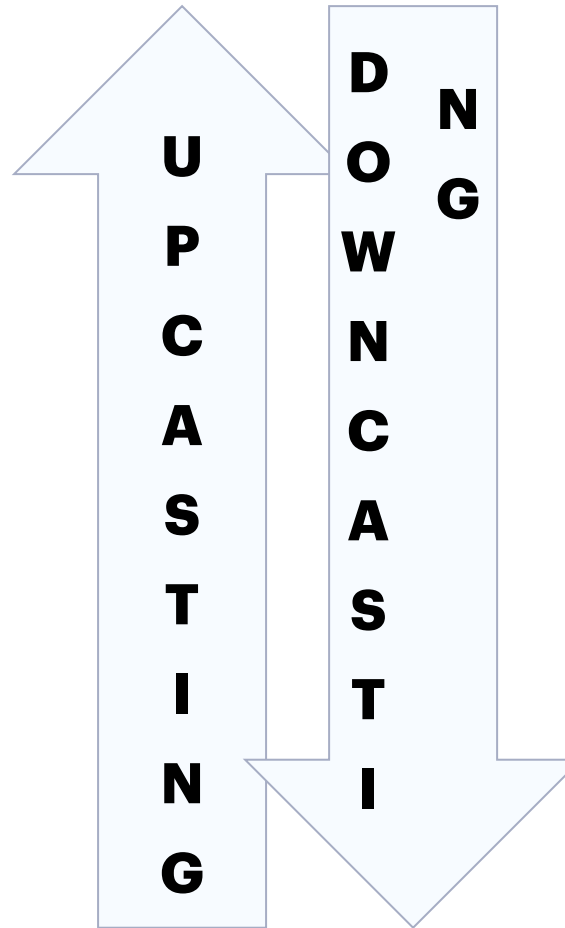
**Example A:**        **double a = 1.5;**  
                      **int b;**  
                      **b = ( int ) a**

- Because an integer is 'narrower' than the double being assigned, it has to be explicitly stated that the assignment is intentional.
- A narrowing conversion will sometimes lose a part of the original value during conversion. For instance, in Example A, variable 'b' will be



# Reference Casting

- Refers to the conversion of a reference data type (an object) to another reference data type



# Java Operators

- *Arithmetic operators* are used to perform mathematical operations on numeric values

**<operand 1> + <operand 2>**  
**<operand 1> - <operand 2>**  
**<operand 1> / <operand 2>**  
**<operand 1> \* <operand 2>**  
**<operand 1> % <operand 2>**

**Addition**  
**Subtraction**  
**Division**  
**Multiplication**  
**Modulo**

**Examples:**

**1 + 1**

**A \* (Z + 1)**

# Java Operators

- *Assignment operators* are used to assign values to variables

**<variable> = <expression>;**

**Examples:**

**name = "John Doe";**

**age = 23;**

**Date today = new Date();**

- The following are unary operators
  - prefix
    - *$Y = ++x$ ; Add 1 to  $x$  before being assigned to  $y$*
    - *$Y = --x$ ; Subtract 1 from  $x$  before being assigned to  $y$*

# Java Operators

*Relational operators* are used to perform comparisons  
They always evaluate to either a *true* or *false* (*boolean expression*)

- <Expression 1> < <Expression 2> Expression 1 is less than Expression 2
- <Expression 1 > > <Expression 2> Expression 1 is greater than Expression 2
- <Expression 1 > <= <Expression 2> Expression 1 is less than or equal to Expression 2
- <Expression 1 > >= <Expression 2> Expression 1 is greater than or equal to Expression 2
- <Expression 1> != <Expression 2> Expression 1 is not equal to Expression 2

Example:

```
boolean test = x < y;
```

# Java Operators

- *Logical operators are used to evaluate boolean expressions*

*<Expression 1> && <Expression 2> Short-circuited AND*

*<Expression 1> || <Expression 2> Short-circuited OR*

*<Expression 1> & <Expression 2> bitwise AND (Non Short Circuited AND)*

*<Expression 1> | <Expression 2> bitwise OR (Non Short Circuited OR)*

*<Expression 1> ^ <Expression 2> XOR*

*!(<Expression 2>) NOT*

*Example:*

*boolean test = x >= 1 && x <= 10*

# Activity 2

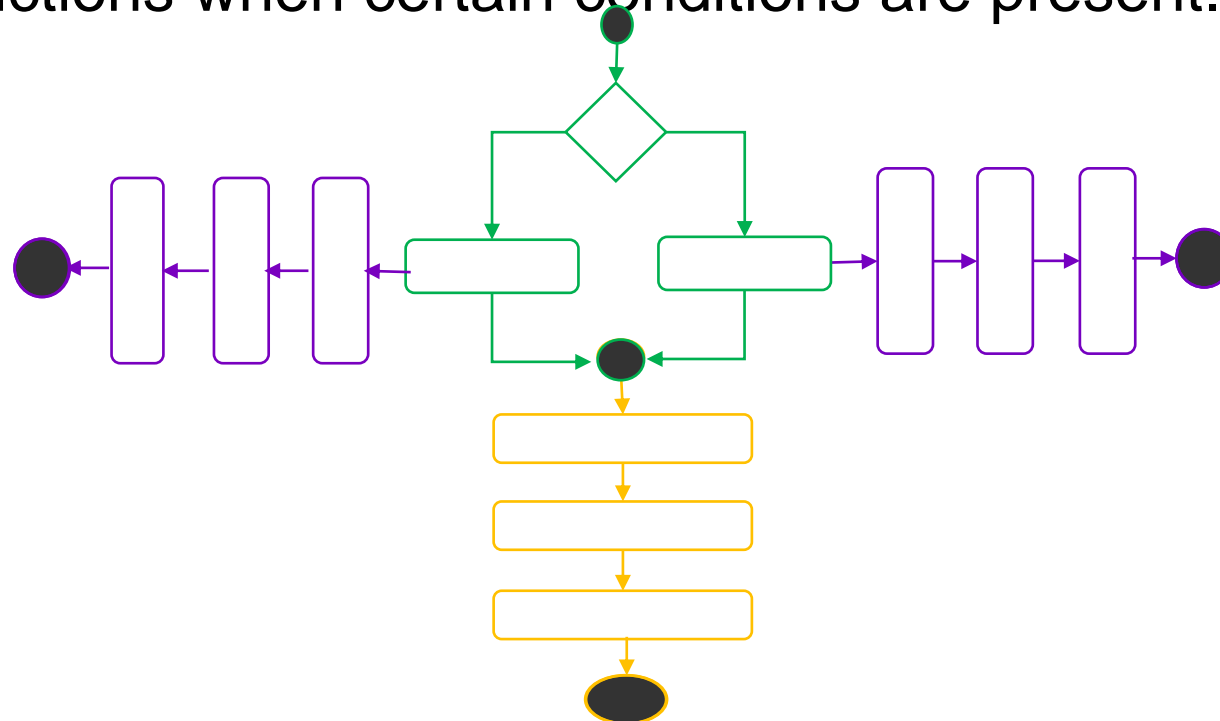
- Open the file 'OperatorActivity.java' in the package `sef.module3.activity`
- Perform the following:
  - Find the difference of the given integers
  - Print the result





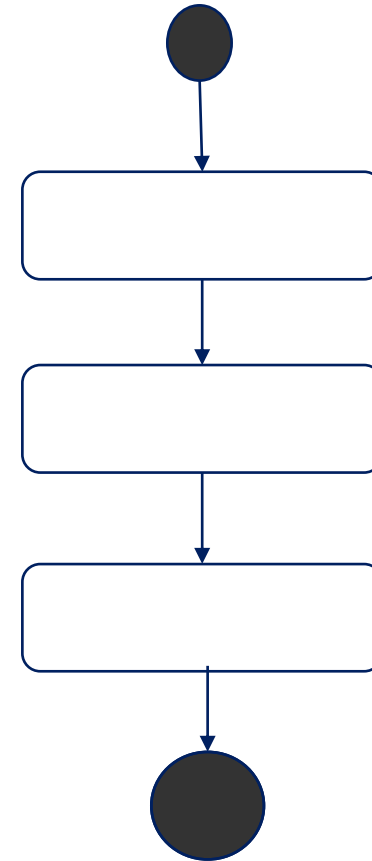
# Flow Control

- A programs 'flow' refers to the order in which statements are executed.
- By default, statements are executed sequentially.
- Flow control statements are used to alter or modify the path of execution of instructions when certain conditions are present.



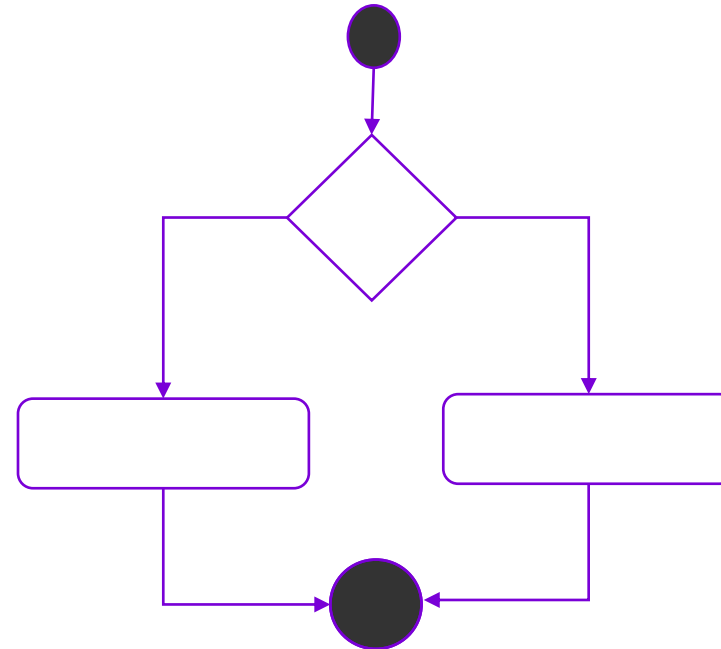
# Types of Flow Control

- **Sequential** statements are executed in the order they are written
- This is the default flow of a program as instructions are executed in the order they appear



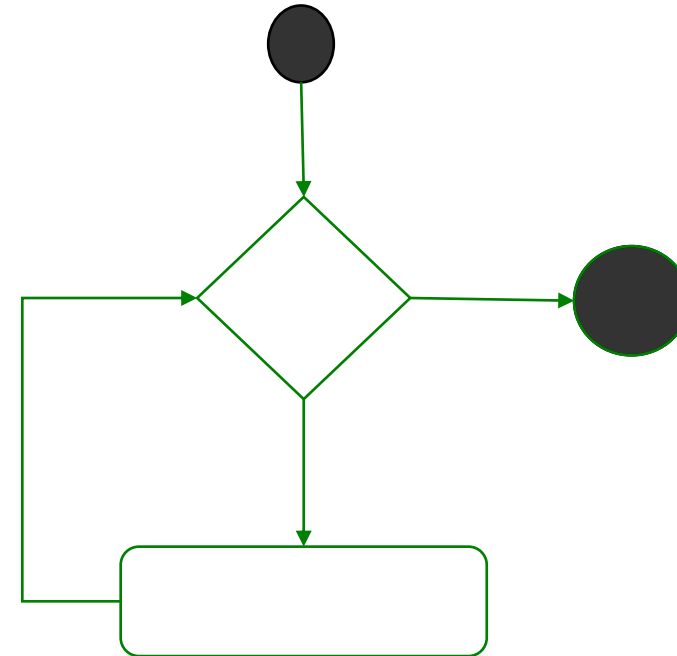
# Types of Flow Control

- Selection structures execute a certain branch based on the results of a boolean condition
- This flow is useful in decision-making scenarios and can be implemented by using the if-else or switch-case statements



# Types of Flow Control

- Iteration structures execute instructions repeatedly based on a condition
- This type of flow control can be implemented by making use of the different loop statements:
  - for-loop
  - while-loop
  - do-while loop



# If-Else

- The outcome of an if-else performs statement is based on a condition

```
Syntax:  if ( condition ) {  
           // statement(s) to be executed If condition is met  
        } else { // ( else statement is optional )  
           // statement(s) to be executed If condition is NOT met  
        }
```

```
Example: int x = 1;  
        if ( x > 0 ) {  
            System.out.println( "The value of x is greater than zero" );  
        } else {  
            System.out.println( "The value of x is less than or equal to zero" );  
        }
```

# Activity 3

- Open the file 'FindLargest.java' in the package `sef.module3.activity`
- Perform the following:
  - Complete the code. Use if-else statement to find the largest of the two given numbers
  - Print the result



# Switch-Case

- A switch-case allows the program to choose which statement(s) to perform based on a condition

## Syntax:

```
switch (exp) {  
  case val1: // statements here  
    break;  
  case val2: // statements here  
    break;  
  default: // statements here  
}
```

## Example:

```
int x = 1;  
switch (x) {  
  case 1: System.out.println ( "Value of x is 1");  
    break;  
  case 2: System.out.println ( "Value of x is NOT 1");  
    break;  
  default: System.out.println ( "Value of x is NULL");  
}
```

# Activity 4

- Open the file 'NumToWords.java' in the package sef.module3.activity
- Perform the following:
  - Complete the code and print text value of number 5





# For Loop

- A for-loop performs statement(s) repeatedly if a certain condition is satisfied

**Syntax:**

```
for ( init; condition; exp ) {  
    //statements here  
}
```

**Example:**

```
for ( int i = 1; i < 11; i++ ) {  
    System.out.println ( "The value of i is: " + i );  
}
```

# Activity 5

- Open the file 'AddWholeNum.java' in the package `sef.module3.activity`
- Perform the following:
  - Complete the code and write a for loop to add all whole numbers from 1 to 50
  - Print the result



# Activity 6

- Open the file 'MultiplicationTable.java' in the package sef.module3.activity
- Perform the following:
  - Complete the code and write for loops to print multiplication table from 1 to 10



# While Loop

- while() performs statements repeatedly while a condition remains true

**Syntax:**

```
while ( condition ) {  
    //statements here  
}
```

**Example:**

```
int x = 1;  
while ( x < 11) {  
    System.out.println ( "The value of x is: " + x );  
}
```



Refer to the WhileLoopSample.java sample code.

# Activity 7

- Open the file 'PrintNumWithWhile.java' in the package sef.module3.activity
- Perform the following:
  - Complete the code and write a while loop to print all even numbers less than 100



# Do-While Loop

- do-while() performs statements repeatedly (at least once) while condition remains true

**Syntax:**

```
do {  
    //place statements here  
} while ( condition )
```

**Example:**

```
int x = 1;  
do {  
    System.out.println ( "The value of x is: " + x );  
} while ( x < 11)
```



Refer to the WhileLoopSample.java sample code.

# Arrays

- An array is a sequence of either objects or primitives, all of the same type under one identifier name
- Basic array declaration syntax:

**Syntax:**

```
<data_type> [ ] <variable_name>;  
<data_type> <variable_name> [ ];  
<data_type> [ ] <variable_name> [ ];
```

**Examples :**

```
int [ ] myIntegerArray;  
String myStringArray [ ];  
char [ ] myCharArray [ ];
```

# Arrays: Manipulation

- *Values* in arrays can be accessed and manipulated by their *index*
- An array's index always start with 0

```
String[] anotherStringArray = {"Hello", "There", "How", "Are", "You"};  
System.out.println("The first string is " + anotherStringArray[0]);
```

ARRAY					
INDEX	0	1	2	3	4
VALUE	"Hello"	"There"	"How"	"Are"	"You"



# Activity 8

- Open the file 'FindInArray.java' in the package `sef.module3.activity`
- Perform the following:
  - Complete the code to find the smallest number in the given array



# Arrays: Multi-Dimensional Arrays

- A multi-dimensional array can simply be thought of as arrays of arrays
- Each pair of square brackets ( [ ] ), represent one dimension

**Syntax:**                   <data type>[ ][ ] <variable name>;  
                              <data type>[ ][ ][ ] <variable name>;

**Example:**                               int[ ][ ] aGridOfIntegers;  
                              int[ ][ ][ ] aCubeOfIntegers;



Refer to the MultiDimensionalArraySample.java sample code.

# Arrays: Manipulation

- Multi-dimensional arrays can be accessed the same way
- Each bracket pair represents one dimension of the array

```
char[][] anotherCharArray = {{a,b,c},{d,e,f},{g,h,i}};  
System.out.println("Accessing center of grid" + anotherCharArray[1][1]);
```

	0	1	2
0	a	b	c
1	d	e	f
2	g	h	i



Refer to the MultiDimensionalArraySample.java sample code.

# Methods

- A method is a collection of one or more statements that performs a specific task
- Basic syntax of a Java method declaration:

```
Syntax:      <return_type> <identifier_name> ( <parameter(s)> ) {  
                //method implementation here  
                return <return_value>;  
            }
```

- A method's *signature* is a combination of a method's *name* and *parameters* that uniquely identify a method

# Method Declaration

## **greet()**

No parameters with a 'void' return type. This means the method does not return any value

```
package sef.module3.sample;
```

```
public class MethodSample {
```

```
    public void greet(){  
        System.out.println("Hello!");  
    }
```

## **greet(String)**

- You can pass parameters to a method. These are considered local variables
- It has the same name as the previous method, but different parameters
- Note a 'static' modifier. This means this method is a class method and can be called without an object reference

```
    public static void greet(String name){  
        System.out.println("Hello " + name +  
        "!");  
    }
```

```
    public int sum(int x, int y) {  
        return x + y;  
    }
```

```
}
```

## **thirdMethod**

This method has a *int* return type. This requires the method to have a 'return' statement that returns an integer value.

# Calling Methods

- To 'call' a method, use the method's name and pass the appropriate number and type of parameters according to the method's signature.
- When calling a method, flow control will 'jump' to that method.
- When the method finishes, flow control will return to the point where the method was called.
- Instance methods are attached to an instance of an object. These will need to be called through an object reference.
- Static methods are attached to a class, and can be called via its class name.

# Parameter Passing

- Parameters in Java are Passed By Value
  - Passing Primitive Data Type Arguments
    - A copy of the value is passed to the method.
    - Any changes to the value exists only within the scope of the method.
    - When the method returns, any changes to the value are lost. The original value remains.
  - Passing Reference Data Type Arguments
    - A copy of the object's reference's value is being passed.
    - The values of the object's fields can be changed inside the method, if they have the proper access level.
    - When the method returns, the passed-in reference still references the same object as before. However, the changes in the object's fields will be

# Method Declaration

Call an instance method through its object

Call class methods statically and pass parameters

Call an instance method that accepts parameters and returns values

```
package sef.module3.sample;

public class MethodSample {

    public void greet(){
        System.out.println("Hello!");
    }

    public static void greet(String name){
        System.out.println("Hello " + name + "!");
    }

    public int sum(int x, int y) {
        return x + y;
    }

    public static void main(String arg[]){
        MethodSample sample = new MethodSample();
        sample.greet();
        greet("Java Teacher");
        MethodSample.greet("Java Student");
        System.out.println("Sum of 1 and 2 is " +
            sample.sum(1, 2));
    }
}
```



# Activity 9

- Open the file 'NumToWordsUsingMethod.java' in the package sef.module3.activity
- Perform the following:
  - Take a look at NumToWords.java in which we wrote code to convert integer numbers into their text value
  - NumToWordsUsingMethod.java will perform the same function using method printWord()
  - Complete the code in



# Activity 10

- Open the file 'Calculator.java' in the package `sef.module3.activity`.
- Notice that method `add()` and `multiply()` are called from inside `main()` method
- Perform following:
  - Complete the code for `add()` and `multiply()` methods
  - Print the Results



# Questions and Comments

- What questions or comments do you have?

