IQRA UNIVERSITY ISLAMABAD

Object Oriented Programming

Lab Manual

Department of Computer Science

Prepared By:

Muhammad Wahab Khan

Junior Lecturer

Lab 01

Introduction to JAVA

Introduction to Java:

- ### Why Java is Important?

Issue with C/C++: Non Portable, non- platform independent

Need of JAVA: Portability and Security

WORA: Write once, run anywhere

- We will study Java as a general-purpose programming language
  - The syntax of expressions and assignments will be similar to that of other high-level languages
  - Details concerning the handling of strings and console output will probably be new

Salient Features of JAVA:

- Java is *platform-independent:* the same program can run on any correctly implemented Java system
- Java is *object-oriented:*
  - Structured in terms of *classes*, which group data with operations on that data
  - Can construct new classes by *extending* existing ones

- Java designed as
  - A *core language* plus
  - A rich collection of *commonly available packages*
- Java can be embedded in Web pages

Objects and Methods

- Java is an *object-oriented programming (OOP)* language
  - Programming methodology that views a program as consisting of *objects* that interact with one another using actions (called *methods*)
  - Objects of the same kind are said to have the same *type* or be in the same *class*

Classes, Objects, and Methods

- A *class* is the name for a type whose values are objects
- *Objects* are entities that store data and take actions
  - Objects of the String class store data consisting of strings of characters
- The actions that an object can take are called *methods*
  - Methods can return a value of a single type and/or perform an action
  - All objects within a class have the same methods, but each can have different data values
- *Invoking* or *calling a method*: a method is called into action by writing the name of the calling object, followed by a dot, followed by the method name, followed by parentheses
  - This is sometimes referred to as *sending a message to the object*
  - The parentheses contain the information (if any) needed by the method
  - This information is called an *argument* (or *arguments*)

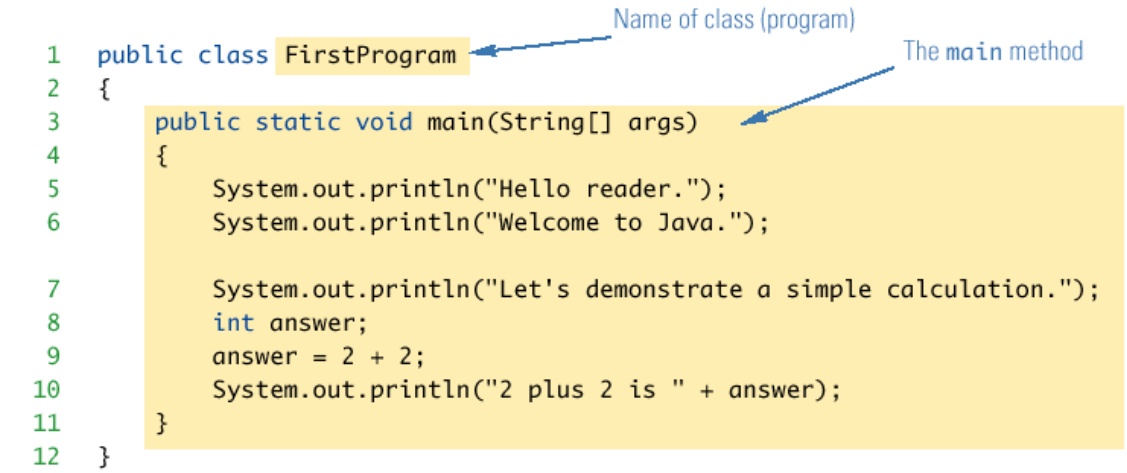
Terminology Comparisons

- Other high-level languages have constructs called procedures, methods, functions, and/or subprograms
  - These types of constructs are called *methods* in Java
  - All programming constructs in Java, including *methods*, are part of a *class*

Java Application Programs

- A Java *application program* or "regular" Java program is a class with a method named main
  - When a Java application program is run, the *run-time system* automatically invokes the method named main
  - All Java application programs start with the main method

Display 1.1 A Sample Java Program




```
1  public class FirstProgram
2  {
3      public static void main(String[] args)
4      {
5          System.out.println("Hello reader.");
6          System.out.println("Welcome to Java.");
7          System.out.println("Let's demonstrate a simple calculation.");
8          int answer;
9          answer = 2 + 2;
10         System.out.println("2 plus 2 is " + answer);
11     }
12 }
```

SAMPLE DIALOGUE 1

```
Hello reader.
Welcome to Java.
Let's demonstrate a simple calculation.
2 plus 2 is 4
```

System.out.println:

- Java programs work by having things called *objects* perform actions
  - System.out: an object used for sending output to the screen
- The actions performed by an object are called *methods*
  - println: the method or action that the System.out object performs
- *Invoking* or *calling* a method: When an object performs an action using a method
  - Also called *sending a message* to the object
  - Method invocation syntax (in order): an object, a dot (period), the method name, and a pair of parentheses
  - Arguments: Zero or more pieces of information needed by the method that are placed inside the parentheses

```
System.out.println("This is an argument");
```

Variable declarations

- Variable declarations in Java are similar to those in other programming languages
  - Simply give the type of the variable followed by its name and a semicolon

```
int answer;
```

Using = and +

- In Java, the equal sign (=) is used as the *assignment operator*
  - The variable on the left side of the assignment operator is *assigned the value* of the expression on the right side of the assignment operator

```
answer = 2 + 2;
```

- In Java, the plus sign (+) can be used to denote addition (as above) or *concatenation*

Using +, two strings can be connected together

```
System.out.println("2 plus 2 is " + answer);
```

Computer Language Levels

- *High-level language*: A language that people can read, write, and understand
  - A program written in a high-level language must be translated into a language that can be understood by a computer before it can be run
- *Machine language*: A language that a computer can understand
- *Low-level language*: Machine language or any language similar to machine language
- *Compiler*: A program that translates a high-level language program into an equivalent low-level language program
  - This translation process is called *compiling*

Byte-Code and the Java Virtual Machine

- The compilers for most programming languages translate high-level programs directly into the machine language for a particular computer
  - Since different computers have different machine languages, a different compiler is needed for each one
- In contrast, the Java compiler translates Java programs into *byte-code*, a machine language for a fictitious computer called the *Java Virtual Machine*
  - Once compiled to *byte-code*, a Java program can be used on any computer, making it very portable
- *Interpreter:* The program that translates a program written in Java byte-code into the machine language for a particular computer when a Java program is executed
  - The interpreter translates and immediately executes each byte-code instruction, one after another

- Translating byte-code into machine code is relatively easy compared to the initial compilation step

Program terminology

- *Code*: A program or a part of a program
- *Source code* (or *source program*): A program written in a high-level language such as Java
  - The input to the compiler program
- *Object code*: The translated low-level program
  - The output from the compiler program, e.g., Java byte-code
  - In the case of Java byte-code, the input to the Java byte-code interpreter

Class Loader

- Java programs are divided into smaller parts called *classes*
  - Each class definition is normally in a separate file and compiled separately
- *Class Loader*: A program that connects the byte-code of the classes needed to run a Java program
  - In other programming languages, the corresponding program is called a *linker*

Syntax and Semantics

- *Syntax*: The arrangement of words and punctuations that are legal in a language, the *grammar rules* of a language
- *Semantics*: The meaning of things written while following the syntax rules of a language

Tip: Error Messages

- *Bug*: A mistake in a program
  - The process of eliminating bugs is called *debugging*
- *Syntax error:* A grammatical mistake in a program
  - The compiler can detect these errors, and will output an error message saying what it thinks the error is, and where it thinks the error is
- *Run-time error:* An error that is not detected until a program is run
  - The compiler cannot detect these errors: an error message is not generated after compilation, but after execution
- *Logic error:* A mistake in the underlying algorithm for a program
  - *The compiler cannot detect these errors, and no error message is generated after compilation or execution, but the program does not do what it is supposed to do*

Naming Constants

- Instead of using "anonymous" numbers in a program, always declare them as named constants, and use their name instead

```
public static final int INCHES_PER_FOOT = 12;

public static final double RATE = 0.14;
```

  - This prevents a value from being changed inadvertently
  - It has the added advantage that when a value must be modified, it need only be changed in one place
  - Note the naming convention for constants: Use all uppercase letters, and designate word boundaries with an underscore character

Comments

- A *line comment* begins with the symbols //, and causes the compiler to ignore the remainder of the line

- This type of comment is used for the code writer or for a programmer who modifies the code

- A *block comment* begins with the symbol pair **/***, and ends with the symbol pair ***/**
  - The compiler ignores anything in between
  - This type of comment can span several lines
  - This type of comment provides documentation for the users of the program

Display 1.8 Comments and a Named Constant

```
1  /**
2   Program to show interest on a sample account balance.
3   Author: Jane Q. Programmer.
4   E-mail Address: janeq@somemachine.etc.etc.
5   Last Changed: September 21, 2004.
6  */
7  public class ShowInterest
8  {
9      public static final double INTEREST_RATE = 2.5;

10     public static void main(String[] args)
11     {
12         double balance = 100;
13         double interest; //as a percent

14         interest = balance * (INTEREST_RATE/100.0);
15         System.out.println("On a balance of $" + balance);
16         System.out.println("you will earn interest of $"
17                                                 + interest);
18         System.out.println("All in just one short year.");
19     }
20 }
```

Although it would not be as clear, it is legal to place the definition of INTEREST_RATE here instead.

SAMPLE DIALOGUE

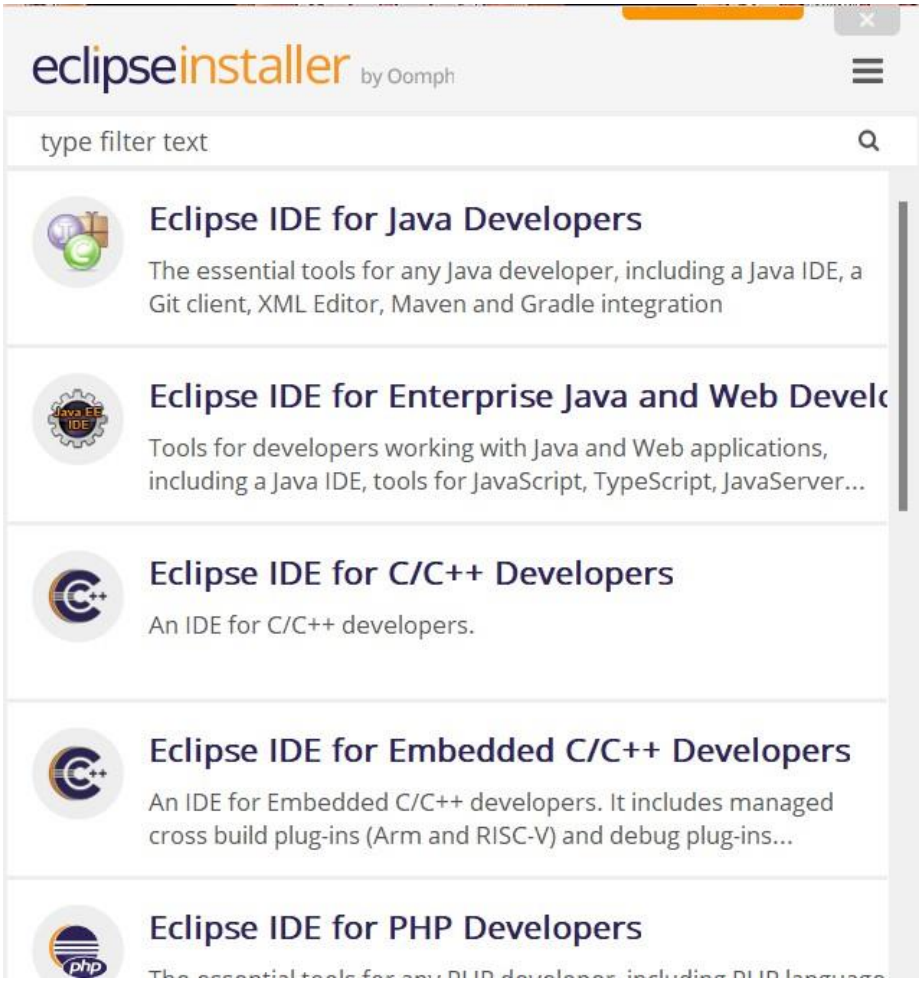
```
On a balance of $100.0
you will earn interest of $2.5
All in just one short year.
```

Eclipse – IDE For Java

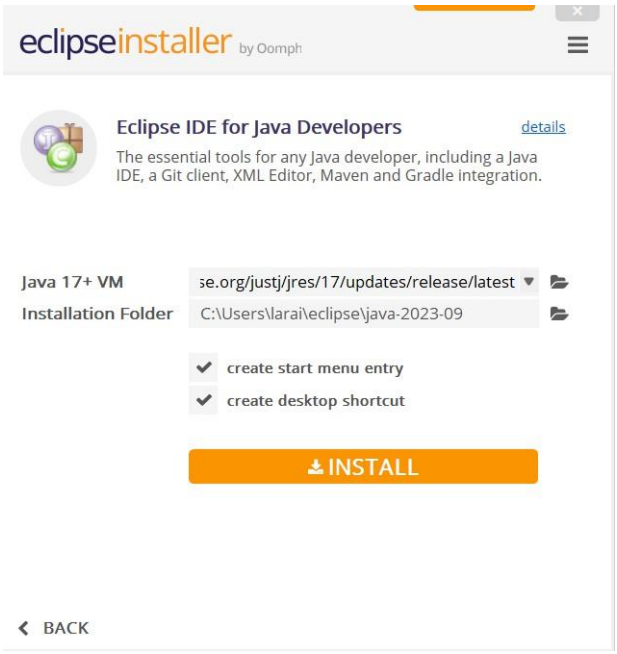
- Eclipse IDE for Java developers is a well-established Java IDE.
- Key Features: Available for Linux, Mac, and Windows
- Support for popular languages like PHP, C++, and JavaScript
- Free and open source
- Supports Java 19
- Install from the link: https://www.eclipse.org/downloads/packages/installer

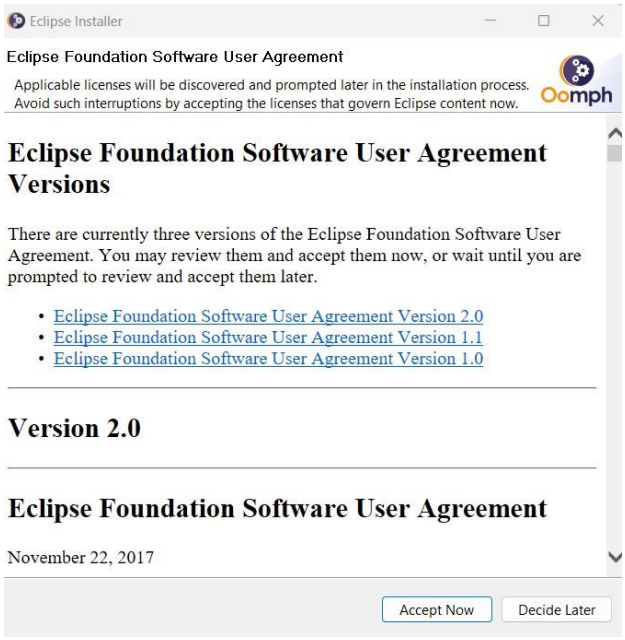
Eclipse – Installation

- After downloading the setup.exe, select the required toolkit: "Eclipse IDE for Java Developers"

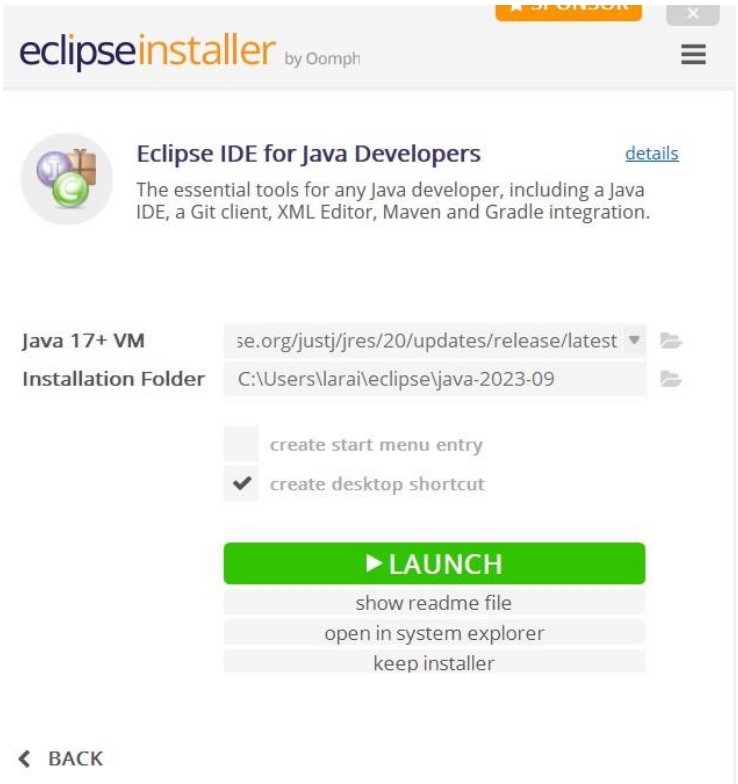


- Select your installation folder and accept the user agreement

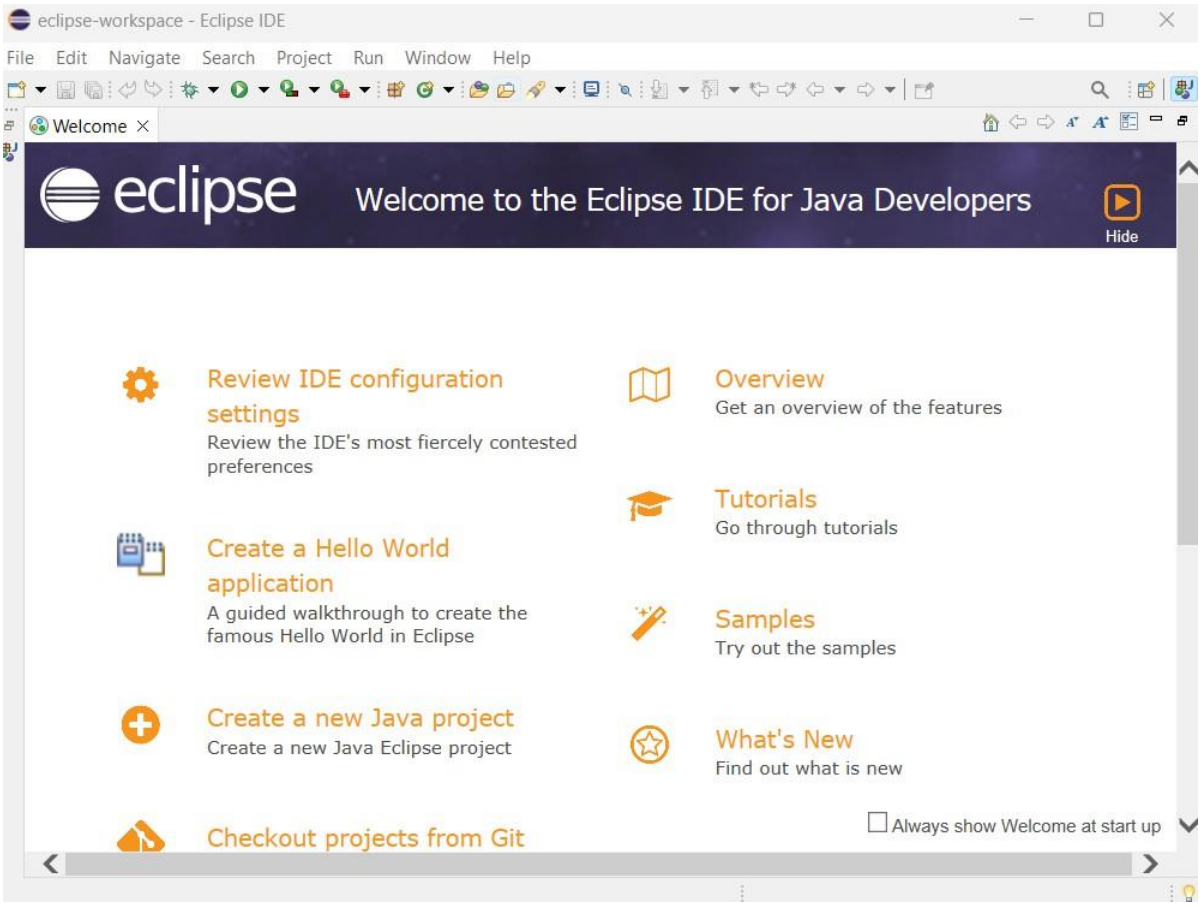




- Launch the Eclipse



- This opens up your workspace for java development



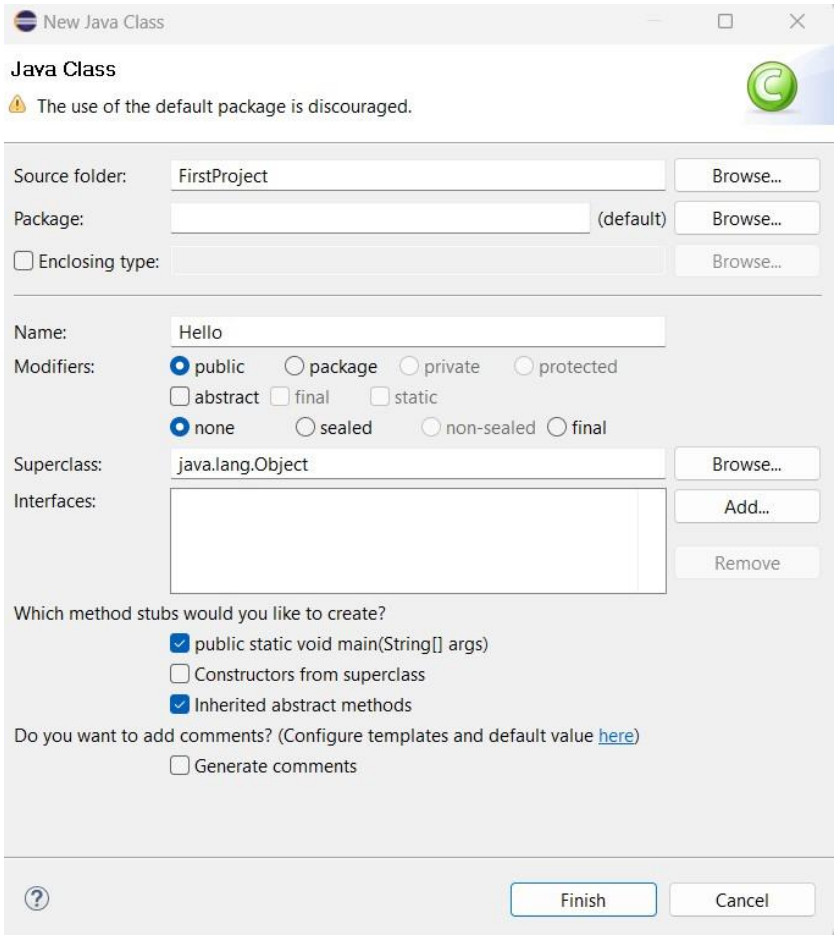
Your First Program in JAVA

Step1: Create a new Java Project

- In Project Name, enter “FirstProject”
- Check “Use Default Location”
- In “JRE”, select "Use default JRE 'jre' and workspace compiler preferences“
- In "Project Layout", check "Use project folder as root for sources and class files".
- In "Module", UNCHECK "Create module-info.java" file.
- Push "Finish" button.

Step 2: Write Hello World Java Program

- In “Package Explorer”, Right Click and select New -> Class
- The new JAVA Class dialog pop ups: keep the settings as follows:



- Write program to display ; hello world

*Hello.java

```
1
2  public class Hello {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6          System.out.println("hello world");
7      }
8
9  }
10
```

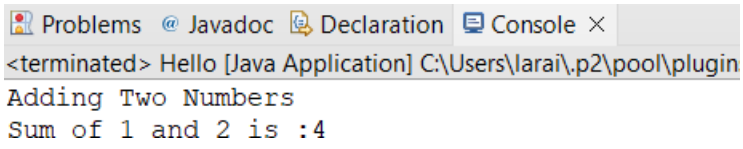
Compile and Execute

- There is no need to *compile* the Java source file in Eclipse explicitly. It is because Eclipse performs the so-called *incremental compilation*, i.e., the Java statement is compiled as and when it is entered.
- To run the program, right-click anywhere on the source file "Hello.java" (or choose "Run" menu) ⇒ Run As ⇒ Java Application.
- The output "Hello, world!" appears on the Console pane (the bottom pane).

Example: To Add Two Numbers

```
1
2  public class Hello {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6          System.out.println("Adding Two Numbers");
7          int ans;
8          ans = 2+2;
9          System.out.println("Sum of 1 and 2 is :" + ans);
10
11     }
12
13 }
14
```

Output:



Input/Output in Java

- Output on Screen: Systems.out.println(“ “)
- Input from User: In Java, we will use the object of Scanner class to take input

```
1  import java.util.Scanner; // import package
2  public class Input {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6          Scanner input = new Scanner(System.in); //creating object of Scanner
7          System.out.print("Enter an integer: ");
8          int number = input.nextInt();   //taking input from user
9          System.out.println("You entered " + number);
10
11         // closing the scanner object
12         input.close();
13
14     }
15
16 }
17
```

Variables and Data Types in JAVA

- A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.
- Variable is a name of memory location. There are three types of variables in java: local, instance and static.

Types of Variables

- **Local Variable:** A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

- **Instance Variable:** A variable declared inside the class but outside the body of the method, is called an instance variable. It is called an instance variable because its value is instance-specific and is not shared among instances.
- **Static variable:** A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory

Example:

```
public class A
{
    static int m=100;//static variable
    void method()
    {
        int n=90;//local variable
    }
    public static void main(String args[])
    {
        int data=50;//instance variable
    }
}//end of class
```

Type Casting in JAVA

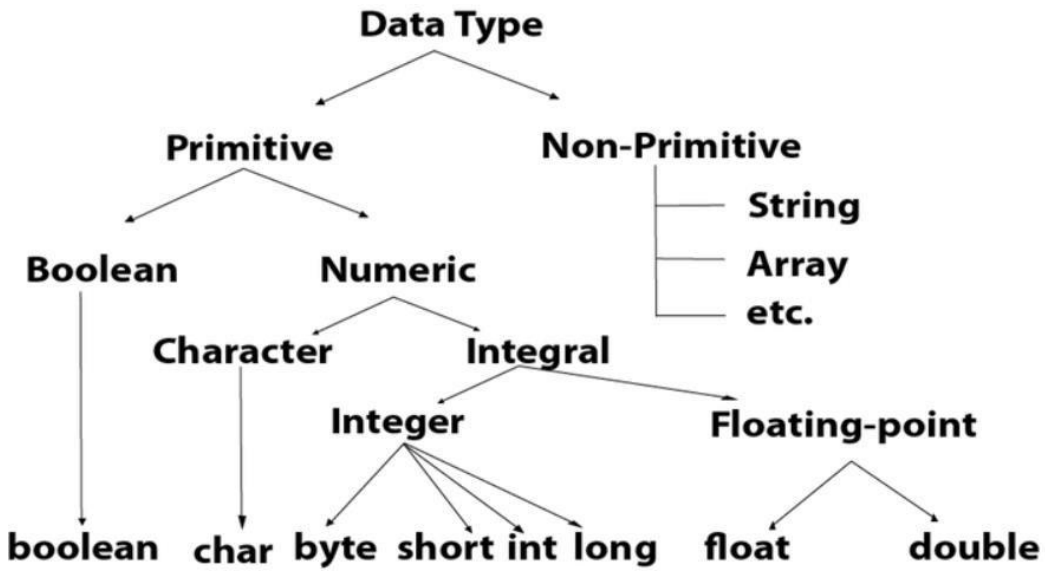
```
2 public class Hello{
3 public static void main(String[] args){
4 int a=10;
5 float f=a;
6 System.out.println(a);
7 System.out.println(f);
8 }}
9
```

```
public class Hello{
        public static void main(String[] args){
        float f=10.5f;
        int a=f;//Compile time error

        System.out.println(f);
        System.out.println(a);
        }}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Type mismatch: cannot convert from float to int

        at Hello.main(Hello.java:5)
```

Data Types in JAVA

- Primitive Data Types
  - Boolean, Char, Byte, Short, int, Long, float and Double
- Non-Primitive Data Types
  - Classes, Interfaces, Arrays, Strings



Data Type	Default Size in Memory
Boolean	1 bit
Char	2 byte
Byte	1 byte
Short	2 byte
Int	4 byte
Long	8 byte
Float	4 byte
Double	8 byte

Operators in JAVA

Operator Type	Category	Precedence
Unary	postfix	*expr++ expr--*
	prefix	*++expr --expr +expr -expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Control Statements

- Decision Making statements
  - if statements (if-else, if-else-if, nested if)
  - switch statement
- Loop statements
  - do while loop
  - while loop
  - for loop
  - for-each loop
- Jump statements
  - break statement
  - continue statement

Lab Tasks:

- Write a Java program to print 'Hello' on screen and your name on a separate line.
- Write a Java program to print the results of the following operations.
  Test Data:
  a. -5 + 8 * 6
  b. (55+9) % 9
  c. 20 + -3*5 / 8
  d. 5 + 15 / 3 * 2 - 8 % 3
- Write a Java program that reads a number in inches and converts it to meters.
  Note: One inch is 0.0254 meter.
- Compute the area of the circle using double variable. ($A=\pi r$^2)
- Write a JAVA Program to display the given pattern using System.out.println

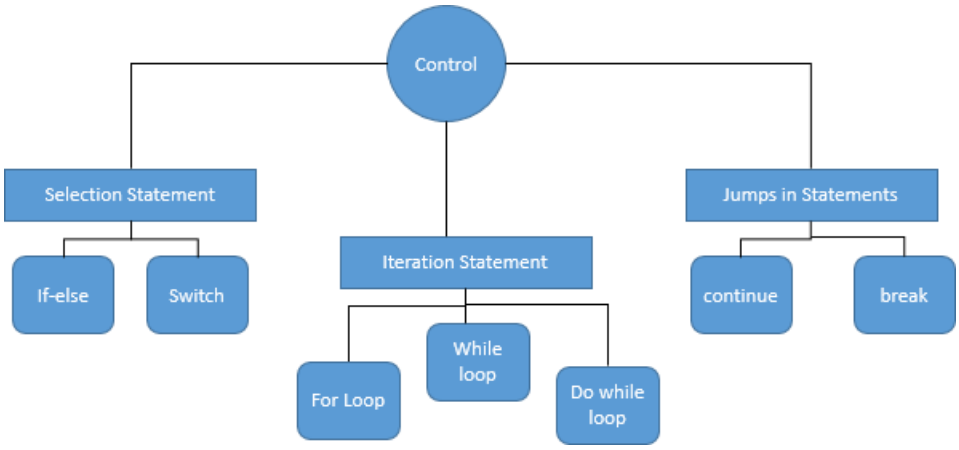
IQRA UNIVERSITY IU
H-9 CAMPUS, ISLAMABAD

Lab 02

Control Statements in JAVA

Control Statements

- The control statements are used to control the flow of the execution of the program.
- In Java Program, control structures can be divided into three parts:



Java Conditions and If Statements

- **IF:** Use if to specify a block of code to be executed, if a specified condition is true
- **ELSE:** Use else to specify a block of code to be executed, if the same condition is false
- **ELSE-IF**: Use else if to specify a new condition to test, if the first condition is false
- **NESTED IF**: Use nested if to specify condition inside if block
- **SWITCH:** Use switch to specify many alternative blocks of code to be executed

IF Statement:

- if (*condition*) {

// block of code to be executed if the condition is true

}

Example:

we test two values to find out if 20 is greater than 18. If the condition is true, print some text:

```
if (20 > 18) {
System.out.println("20 is greater than 18");
}
```

Note that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.

Example:

```
1
2  public class control {
3      public static void main (String[]arg) {
4          int x = 50;
5          int y = 30;
6          if (x>y) {
7              System.out.println("X is Greater than Y");
8          }
9      }
10 }
11
```

Problems Javadoc Declaration Console ×

<terminated> control [Java Application] C:\Users\larai\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v20230801-

```
X is Greater than Y
```

Else Statement

- Use the else statement to specify a block of code to be executed if the condition is false.
- Syntax

if (*condition*) {

// block of code to be executed if the condition is true }

else {

// block of code to be executed if the condition is false }

Example

```
1
2  public class control {
3      public static void main (String[]arg) {
4          int x = 13;
5          if (x%2==0) {
6              System.out.println("X is Even Number");
7          }
8          else
9          {
10             System.out.println("X is Odd Number");
11         }
12     }
13 }
14
```

Problems @ Javadoc Declaration Console ×

<terminated> control [Java Application] C:\Users\larai\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v20230

```
X is Odd Number
```

Else-If

- Use the else if statement to specify a new condition if the first condition is false.

Syntax

if (*condition1*) {

// block of code to be executed if condition1 is true }

else if (*condition2*) {

// block of code to be executed if the condition1 is false and condition2 is true }

else {

// block of code to be executed if the condition1 is false and condition2 is false }

Example

```
1
2  public class control {
3      public static void main (String[]arg) {
4          int marks = 65;
5          if (marks<50) {
6              System.out.println("FAIL");
7          }
8          else if (marks>50 && marks<60) {
9              System.out.println("PASS with average Grade");
10         }
11         else {
12             System.out.println("PASS with Distinction");
13         }
14     }
15 }
16
```

Problems Javadoc Declaration Console ×

<terminated> control [Java Application] C:\Users\larai\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v20230801-2057\jre\

```
PASS with Distinction
```

Nested If-Else

The nested if-else statement executes one if or else if statement inside another if or else if statement

Syntax:

If (Condition 1)

{ Executes if true
}

If (Condition 2)

{ Executes if true

}

Example:

```
1
2  public class control {
3      public static void main (String[]arg) {
4          int X = 30;
5          int Y = 10;
6          if (X==30) {
7              if (Y==10) {
8                  System.out.println("X=30 and Y=10");
9              }
10         }
11
12     }
13 }
14
```

Problems Javadoc Declaration Console ×

<terminated> control [Java Application] C:\Users\larai\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v20230801

```
X=30 and Y=10
```

Switch

The switch statement selects one of many code blocks to be executed

Syntax:

switch (expression){

Case x:

//code block

Break;

Case y:

//code block

Default:

}

Example:

```
1
2  public class control {
3      public static void main (String[]arg) {
4          int day = 4;
5          switch (day) {
6            case 1:
7              System.out.println("Monday");
8              break;
9            case 2:
10             System.out.println("Tuesday");
11             break;
12           case 3:
13             System.out.println("Wednesday");
14             break;
15           case 4:
16             System.out.println("Thursday");
17             break;
18           case 5:
19             System.out.println("Friday");
20             break;
21           case 6:
22             System.out.println("Saturday");
23             break;
24           case 7:
25             System.out.println("Sunday");
26             break;
27         }
28
29     }
30 }
31
```

Problems @ Javadoc Declaration Console ×

<terminated> control [Java Application] C:\Users\larai\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win

```
Thursday
```

Lab Tasks:

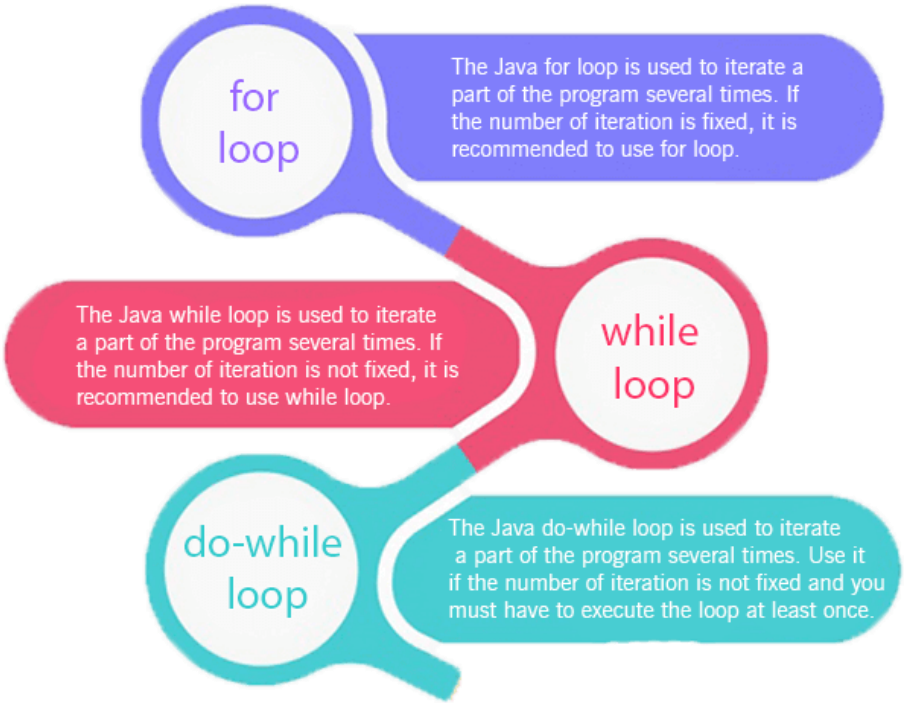
- Write a Java program to display the largest number using nested If statements.
- Write a Java program to display the largest number using nested If statements.
- Create a calculator in JAVA which can perform addition, subtraction, multiplication, and division. Ask the user to enter the operator first then ask the user to enter the first and then second value. Sample Output: Please enter the operator which you want to perform. + Please enter the first value? 5 Please enter the second value. 6 5+6 =11
- Write a program to check whether a triangle is valid or not. The three angles of the triangle are entered through the keyboard. A triangle is valid if the sum of all three angles is equal to 180 degrees
- Write a program that takes a number from the user and displays whether the number is 0, a positive or a negative integer
- Write an if statement for the following situation: If an integer variable currentNumber is a multiple of 8, change its value so that it is now 3 times currentNumber plus 2, otherwise, change its value so that it is now half of currentNumber
- Write a switch statement program to check whether a character is vowel or not

IQRA UNIVERSITY IU
H-9 CAMPUS, ISLAMABAD

Lab 03

Part A – Loops in JAVA

Loops in JAVA

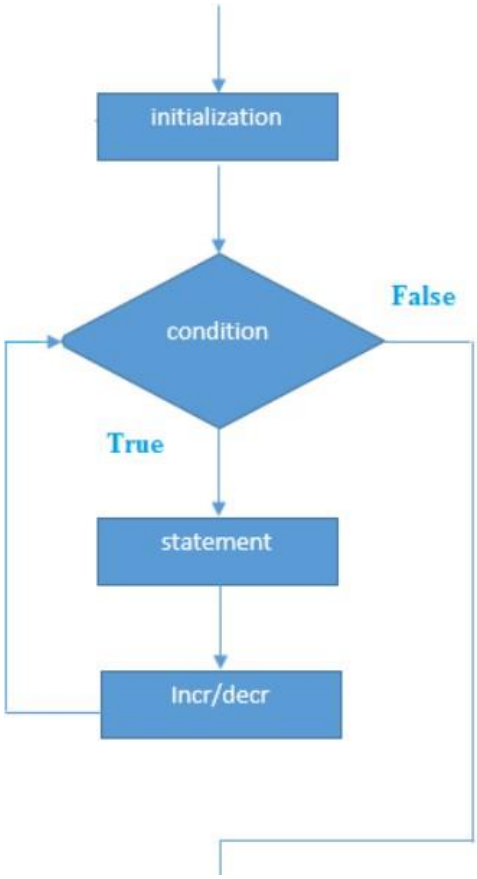


For Loop

for(initialization; condition; increment/decrement){

//statement or code to be executed

}



- **Example**

```
//Java Program to demonstrate the example of for loop
//which prints table of 1
public class ForExample {
public static void main(String[] args) {
     //Code of Java for loop
     for(int i=1;i<=10;i++){
        System.out.println(i);
    }
} }
```

Nested For Loop

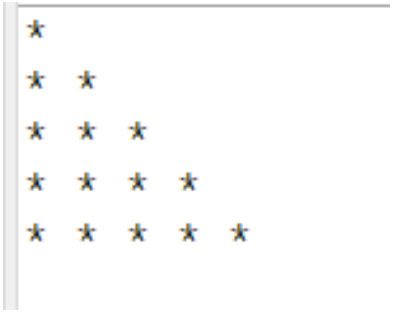
If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

```
public class NestedForExample {

public static void main(String[] args) {

//loop of i

for(int i=1;i<=3;i++){

//loop of j

   for(int j=1;j<=3;j++){

      System.out.println(i+" "+j);

}//end of i

}//end of j

}

}
```

Pyramid Example:

```
public class PyramidExample {

public static void main(String[] args) {

for(int i=1;i<=5;i++){

for(int j=1;j<=i;j++){

      System.out.print("* ");

}

System.out.println();//new line

}
```

} }



JAVA For each loop

- The for-each loop is used to traverse array or collection in Java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.
- It works on the basis of elements and not the index. It returns element one by one in the defined variable.
- **Syntax:**

for(data_type variable : array_name){

//code to be executed

}

Example

```
//Java For-each loop example which prints the
//elements of the array
public class ForEachExample {
public static void main(String[] args) {
    //Declaring an array
    int arr[]={12,23,44,56,78};
    //Printing array using for-each loop
```

```
    for(int i:arr){
        System.out.println(i);
    }
}
}
```

JAVA Labelled For Loop

- We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful while using the nested for loop as we can break/continue specific for loop.
- **Syntax:**

```
labelname:
for(initialization; condition; increment/decrement){
//code to be executed
}
```

Example

```
//A Java program to demonstrate the use of labeled for loop
public class LabeledForExample {
public static void main(String[] args) {
    //Using Label for outer and for loop
    aa:
        for(int i=1;i<=3;i++){
            bb:
                for(int j=1;j<=3;j++){
                    if(i==2&&j==2){
```

```
                    break aa;
                }
                System.out.println(i+" "+j);
            }
        }
}
}
```

JAVA While loop

- The Java *while loop* is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.
- The while loop is considered as a repeating if statement. If the number of iteration is not fixed, it is recommended to use the while loop.
- **Syntax:**

  while (condition){

  //code to be executed

  Increment / decrement statement

  }

Example

```
public class WhileExample {
public static void main(String[] args) {
   int i=1;
   while(i<=10){
      System.out.println(i);
   i++;  }}}
```

JAVA Infinite while loop

- If you pass **true** in the while loop, it will be infinitive while loop.
- **Syntax:**

```
while(true){
//code to be executed
}
```

Example

```
public class WhileExample2 {
public static void main(String[] args) {
 // setting the infinite while loop by passing true to the condition
   while(true){
      System.out.println("infinitive while loop");
   }
}
}
```

JAVA Do While Loop

- The Java *do-while loop* is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop.
- Java do-while loop is called an **exit control loop**. Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body. The Java *do-while loop* is executed at least once because condition is checked after loop body.
- **Syntax:**

```
do{
//code to be executed / loop body
//update statement
}while (condition);
```

Example

```
public class DoWhileExample {
public static void main(String[] args) {
    int i=1;
    do{
        System.out.println(i);
    i++;
    }while(i<=10);
}
}
```

IQRA UNIVERSITY IU
H-9 CAMPUS, ISLAMABAD

Part B – Object & Classes in JAVA

Naming Conventions:

Identifier Type	Naming Rules	Examples

Class	It should start with the uppercase letter.
	It should be a noun such as Color, Button, System, Thread, etc.
	Use appropriate words, instead of acronyms.
		public class **Employee**
		{
		//code snippet
		}

Method	It should start with lowercase letter.
	It should be a verb such as main(), print(), println().
	If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
		class Employee
		{
		// method
		void **draw()**
		{
		//code snippet
		}
		}

Variable	It should start with a lowercase letter such as id, name.
	It should not start with the special characters like & (ampersand), $ (dollar), _ (underscore).
	If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
		class Employee
		{
		// variable
		int **id**;
		//code snippet
		}

	Avoid using one-character variables such as x, y, z.	

Constant	It should be in uppercase letters such as RED, YELLOW.
	If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY.
	It may contain digits but not as the first letter.
		class Employee
		{
		//constant
		static final int **MIN_AGE** = 18;
		//code snippet
		}

Object

- An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code.
- **Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.
- An object has three characteristics:
    - **State:** represents the data (value) of an object.
    - **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
    - **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

Class

- *Collection of objects* is called class. It is a logical entity.
- A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.
- A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.
- Syntax to declare a class:

```
class <class_name>{
    field;
    method;
}
```

JAVA terminologies

- **Instance Variable in JAVA**: A variable that is created inside the class but outside the method is known as an instance variable. The instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.
- **Method in JAVA**: In Java, a method is like a function that is used to expose the behavior of an object.
- **New Keyword in JAVA**: The new keyword is used to allocate memory at runtime

Example – Main inside the class

```
//Java Program to illustrate how to define a class and fields
//Defining a Student class.
class Student{
 //defining fields
 int id;//field or data member or instance variable
 String name;
 //creating main method inside the Student class
 public static void main(String args[]){
  //Creating an object or instance
  Student s1=new Student();//creating an object of Student
  //Printing values of the object
  System.out.println(s1.id);//accessing member through reference variable
  System.out.println(s1.name);
 }
}
```

Example – Main outside the class

```
//Java Program to demonstrate having the main method in
//another class
//Creating Student class.
class Student{
 int id;
 String name;
}
```

```
//Creating another class TestStudent1 which contains the main method
class TestStudent1{
 public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Initialize Object

- Initializing an object means storing data into the object.
- 3 ways to initialize object
  - By reference Variable
  - By method
  - By constructor

Example – Initialization Through reference

```
class Student{
 int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
```

```
 s1.name="Sonoo";
 System.out.println(s1.id+" "+s1.name);//printing members with a white space
 }
}
```

Example – Initialization Through reference

```
class Student{
 int id;
 String name;
}
class TestStudent3{
 public static void main(String args[]){
  //Creating objects
  Student s1=new Student();
  Student s2=new Student();
  //Initializing objects
  s1.id=101;
  s1.name="Sonoo";
  s2.id=102;
  s2.name="Amit";
  //Printing data
  System.out.println(s1.id+" "+s1.name);
  System.out.println(s2.id+"  "+s2.name);
 }
}
```

Example – Initialization Through Method

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Anonymous Object

- Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

- If you have to use an object only once, an anonymous object is a good approach.

Example:

```
class Calculation{
 void fact(int  n){
 int fact=1;
  for(int i=1;i<=n;i++){
   fact=fact*i;
  }
 System.out.println("factorial is "+fact);
}
public static void main(String args[]){
 new Calculation().fact(5);//calling method with anonymous object
}
}
```

Creating Multiple Objects By One Type

Same as we do in primitive data types

```
//Java Program to illustrate the use of Rectangle class which
//has length and width data members
class Rectangle{
 int length;
 int width;
 void insert(int l,int w){
  length=l;
```

```
 width=w;
 }
 void calculateArea(){System.out.println(length*width);}
}
class TestRectangle2{
 public static void main(String args[]){
  Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
  r1.insert(11,5);
  r2.insert(3,15);
  r1.calculateArea();
  r2.calculateArea();
}
}
```

Lab Tasks

- Run all Example Programs of Part 2, and verify output.
- Write a JAVA Program for a banking system :
- Class Account{
  fields// name, account no, amount

  method//void insert()

  Method()void deposit()

  Method()void withdraw()

  Method ()void checkbalance()

  Method() void display values of object

  }
- Create test class to create object, deposit and withdraw amount.

Lab No 04

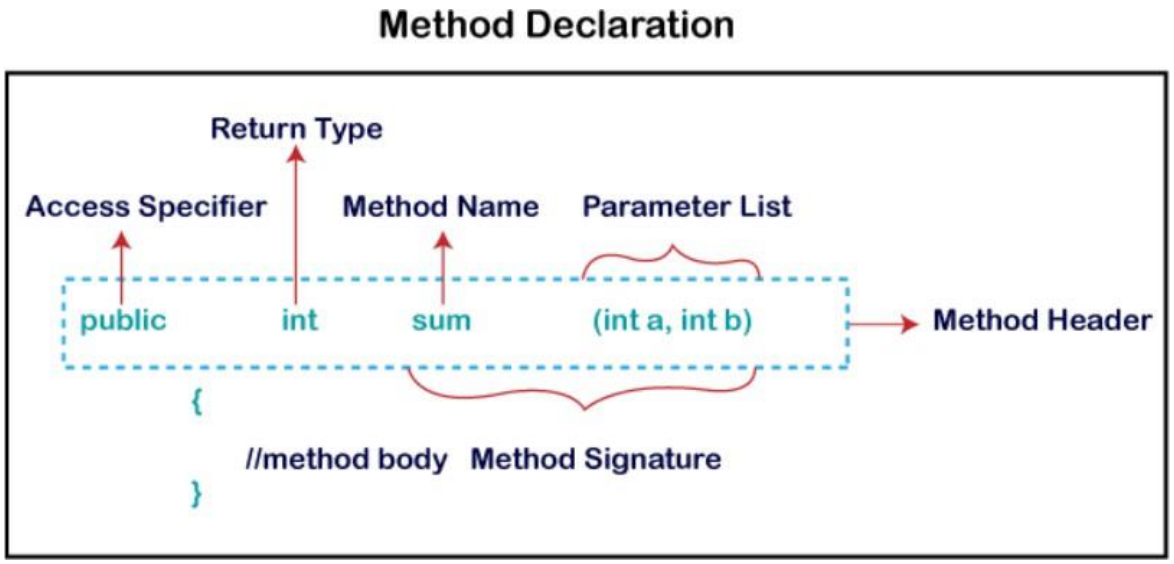
Methods in JAVA

- ## JAVA Method

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code. We write a method once and use it many times. We do not require to write code again and again. It also provides **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

- ## Method Declaration

The method declaration provides information about method attributes, such as visibility, return type, name, and arguments. It has six components that are known as **method headers.**



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.
- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use the void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name. Examples:

a) **Single-word method name:** sum(), area()
b) **Multi-word method name:** areaOfCircle(), stringComparision()

It is also possible that a method has the same name as another method name in the same class, it is known as **method overloading**

Parameter List: It is the list of parameters separated by a comma and enclosed in a pair of parentheses. It contains the data type and variable name. If the method has no parameter, leave the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within a pair of curly braces.

- **Types of Methods**
  - Pre-defined method
  - User-defined method

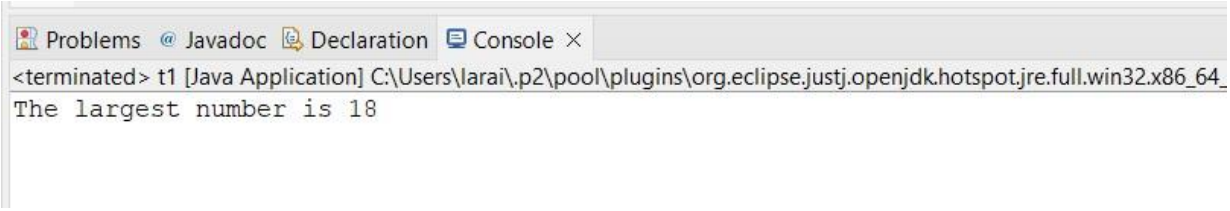
Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

Example:

```
1
2 public class t1 {
3 public static void main (String[]arg) {
4     System.out.println("The largest number is " +Math.max(10, 18));
5 }
6 }
7
```

Output:



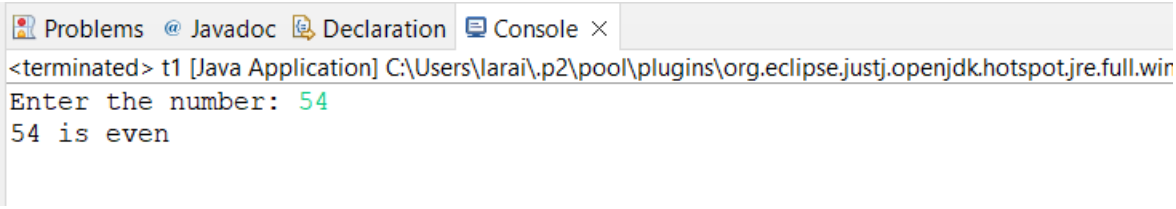
User-defined Method

The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement**.**

Example:

```java
import java.util.Scanner;
public class t1 {


    public static void main (String args[])
    {

    Scanner scan=new Scanner(System.in);
    System.out.print("Enter the number: ");

    int num=scan.nextInt();

    findEvenOdd(num);
    }

    public static void findEvenOdd(int num)
    {

    if(num%2==0)
    System.out.println(num+" is even");
    else
    System.out.println(num+" is odd");

    }
}
```

Output:



TASK: Write a JAVA method to Add two numbers. Take input from user and call the method in Main.

Static Method

A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword **static** before the method name.

The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name.

```
1  import java.util.Scanner;
2  public class t1 {
3
4      public static void main(String[] args)
5      {
6      show();
7      }
8      static void show()
9      {
10     System.out.println("It is an example of static method.");
11     }
12     }
13
14
15
16
17
18
```

Instance Method

The method of the class is known as an **instance method**. It is a **non-static** method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class.

```java
public class t1 {

    public static void main(String[] args)
    {
        //Creating an object of the class
        t1 obj = new t1();
        //invoking instance method
        System.out.println("The sum is: "+obj.add(12, 13));
        }
        int s;
        //user-defined method because we have not used static keyword
        public int add(int a, int b)
        {
        s = a+b;
        //returning the sum
        return s;
        }
}
```

Exercise:

- Write a JAVA method to find the smallest number of three. Numbers should be entered by the user. Hint (Use Math. min)
- Write a JAVA method to find the average of three numbers
- Write a Java method to check whether a year (integer) entered by the user is a leap year or not (A year is a leap year if the year is a multiple of 400 or the year is a multiple of 4 and not a multiple of 100.)
- Write a Java method to count the number of digits in an integer with the value 2. The integer may be assumed to be non-negative.

Submission Instructions:

Take your code and output screenshots and submit them on Blackboard.

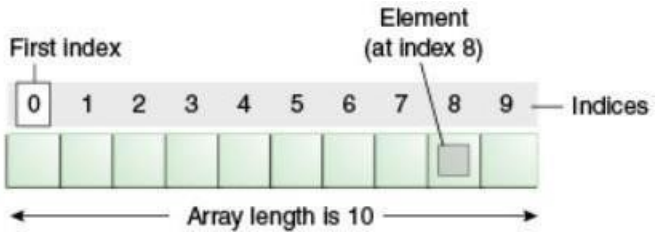
IQRA UNIVERSITY IU
H-9 CAMPUS, ISLAMABAD

Lab No 05

Arrays:

Normally, an array is a collection of similar type of elements which has a contiguous memory location. **Java array** is an object that contains elements of a similar data type.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on the 1st index, and so on.



Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, a collection framework is used in Java which grows automatically.

Types of Array in Java

There are two types of arrays.

- Single Dimensional Array
- Multidimensional Array

Example of Java Array

```
//Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Declaration, Instantiation, and Initialization of Java Array

```
//Java Program to illustrate the use of declaration, instantiation
//and initialization of Java array in a single line
class Testarray1{
public static void main(String args[]){
int a[]={33,3,4,5};//declaration, instantiation and initialization
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Accessing Array Elements:

1- Write a program to access array elements.

```
class ArrayAccessElement {

public static void main(String[] args) {

  // create an array

  int[] age = {12, 4, 5, 2, 5};

  // access each array elements

  System.out.println("Accessing Elements of Array:");

  System.out.println("First Element: " + age[0]);

  System.out.println("Second Element: " + age[1]);

  System.out.println("Third Element: " + age[2]);

  System.out.println("Fourth Element: " + age[3]);

  System.out.println("Fifth Element: " + age[4]);  }  }
```

2- Use Loop to Access Elements of Array

```
class Main {
 public static void main(String[] args) {
   // create an array
   int[] age = {12, 4, 5};
   // loop through the array
   // using for loop
System.out.println("Using for Loop:");
for(int i = 0; i < age.length; i++) {
System.out.println(age[i]);  } } }
```

3- Use For-Each Loop to access elements of Array

```
class Main {
 public static void main(String[] args) {
   // create an array
   int[] age = {12, 4, 5};
   // loop through the array
   // using for loop
   System.out.println("Using for-each Loop:");
   for(int a : age) {
     System.out.println(a); }}}
```

Passing Array to a Method in Java

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see the simple example to get the minimum number of an array using a method.

```
class Testarray2{
static void min(int arr[]){
int min=arr[0];
for(int i=1;i<arr.length;i++)
 if(min>arr[i])
  min=arr[i];
System.out.println(min);  }
public static void main(String args[]){
int a[]={33,3,4,5};
min(a);//passing array to method
```

Anonymous Array in JAVA

```
//Java Program to demonstrate the way of passing an anonymous array
//to method.
public class TestAnonymousArray{
//creating a method which receives an array as a parameter
static void printArray(int arr[]){
for(int i=0;i<arr.length;i++)
System.out.println(arr[i]);
}

public static void main(String args[]){
printArray(new int[]{10,22,44,66});//passing anonymous array to method
}}
```

IQRA UNIVERSITY IU
H-9 CAMPUS, ISLAMABAD

2D- Arrays in JAVA

Write a program to print 2d Array

```
class TwoDArray {
  public static void main(String args[]) {
    int twoD[][] = new int[4][5]; // creating an array
    int i, j, k = 0;
    for (i = 0; i < 4; i++)
      for (j = 0; j < 5; j++) {
      twoD[i][j] = k;
        k++;
      }
    for (i = 0; i < twoD.length; i++) {
      for (j = 0; j < twoD[i].length; j++)
      System.out.print(twoD[i][j] + " ");
      System.out.println();  }  } }
```

Write a program to print 2d diagonal Array

```
// Manually allocate differing size second dimensions.
class TwoDAgain {
  public static void main(String args[]) {
    int twoD[][] = new int[4][];
    twoD[0] = new int[1];
    twoD[1] = new int[2];
    twoD[2] = new int[3];
    twoD[3] = new int[4];
    int i, j, k = 0;
    for (i = 0; i < 4; i++)
      for (j = 0; j < i + 1; j++) {
        twoD[i][j] = k;
        k++;
      }
    for (i = 0; i < 4; i++) {
      for (j = 0; j < i + 1; j++)
        System.out.print(twoD[i][j] + " ");
      System.out.println(); } } }
```

Addition of two matrices in JAVA

```
class Testarray5{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};
//creating another matrix to store the sum of two matrices
int c[][]=new int[2][3];
//adding and printing addition of 2 matrices
for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+"     ");     }
System.out.println();//new line  }}}
```

3D – Array of size 3*4*5

```
// Demonstrate a three-dimensional array.
class ThreeDMatrix {
    public static void main(String args[]) {
        int threeD[][][] = new int[3][4][5];
        int i, j, k;
        for (i = 0; i < 3; i++)
            for (j = 0; j < 4; j++)
                for (k = 0; k < 5; k++)
                    threeD[i][j][k] = i * j * k;
        for (i = 0; i < 3; i++) {
            for (j = 0; j < 4; j++) {
                for (k = 0; k < 5; k++)
                    System.out.print(threeD[i][j][k] + " ");
                System.out.println();
            }
            System.out.println(); } } }
```

IQRA UNIVERSITY IU
H-9 CAMPUS, ISLAMABAD

Tasks:

1- Practice all the above example tasks.

2- Write a JAVA program to find the product of all the elements of a 2D array.

3- Write a program that creates a matrix of size 10 by 10 (10 Columns, and 10 Rows). The program should ask the user to enter values in each matrix element. Then the program should display the left-diagonal elements of the matrix.

4- Write a program that creates a matrix of 3 by 3 (3 rows, and 3 columns). Get input values from the user for the complete matrix. Then, the program should determine whether the matrix is a "Zero" matrix (all elements are zero) or not.

5- Write a program that creates a matrix of 4x4 (4 rows, and 4 columns). Get input values from the user for the complete matrix. The program should calculate and print the sums of each column.

6- Write a JAVA program to find the index of a 1D array element. Your program should contain a method that an integer array and the integer number to search as arguments and return the index of the number.

7- Write a JAVA program to find the duplicate values within a 1D array.

8- You are given an array of integers where all the elements are the same as their index in the array, except for one element. Your task is to find and display the element that has a different index. Write a Java program that takes an array of integers as input and outputs the element in the array that does not match its index.

Submission Instructions:

Submit a pdf file which should contain your code and output screenshots.

IQRA UNIVERSITY IU
H-9 CAMPUS, ISLAMABAD

Lab 06

Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling the constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such cases, the Java compiler provides a default constructor by default.

Note: It is called a constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because the Java compiler creates a default constructor if your class doesn't have any.

Difference between constructor and method in Java

JAVA Constructor	JAVA Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be the same as the class name.	The method name may or may not be same as the class name.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type

Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public, or default constructors in Java.

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor is called a "Default Constructor" when it doesn't have any parameters.

Syntax of default constructor:

```
<class_name>(){}
```

Example of default constructor:

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
1
2  public class Bike {
3
4          //creating a default constructor
5          Bike()
6          {
7              System.out.println("Bike is created");
8          }
9          //main method
10         public static void main(String args[]){
11         //calling a default constructor
12         Bike b=new Bike();
13         }
14         }
15
16
```

Problems Javadoc Declaration Console ×

<terminated> Bike [Java Application] C:\Users\larai\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win

```
Bike is created
```

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

Purpose of a default constructor

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

```
1
2  public class student {
3      int id;
4      String name;
5      //method to display the value of id and name
6      void display(){System.out.println(id+" "+name);}
7
8      public static void main(String args[]){
9      //creating objects
10     student s1=new student();
11     student s2=new student();
12     //displaying values of the object
13     s1.display();
14     s2.display();
15     }
16 }
```

Problems @ Javadoc Declaration Console ×

<terminated> student [Java Application] C:\Users\larai\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full

```
0 null
0 null
```

Explanation: In the above class, we are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of the Student class that has two parameters. We can have any number of parameters in the constructor.

```
public class student {
     int id;
     String name;
    //creating a parameterized constructor
    student(int i,String n){
    id = i;
    name = n;
    }
    //method to display the values
    void display()
    {
        System.out.println(id+" "+name);
        }

    public static void main(String args[]){
    //creating objects and passing values
    student s1 = new student(124,"Ali");
    student s2 = new student(341,"Ayesha");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
```

Problems Javadoc Declaration Console ×

<terminated> student [Java Application] C:\Users\larai\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.

```
124 Ali
341 Ayesha
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without a return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

```java

public class student {
     int id;
     String name;
     int age;
//creating a parameterized constructor  (2 arguments)
    student(int i,String n){
    id = i;
    name = n;
    }
//creating a parameterized constructor  (3 arguments)
    student(int i,String n,int a){
    id = i;
    name = n;
    age = a;
    }

    //method to display the values
    void display()
    {
        System.out.println(id+" "+name+" "+age);
        }

    public static void main(String args[]){
    //creating objects and passing values
    student s1 = new student(124,"Ali");
    student s2 = new student(341,"Ayesha",27);
    //calling method to display the values of object
    s1.display();
    s2.display();
```

Problems Javadoc Declaration Console

```
<terminated> student [Java Application] C:\Users\larai\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v2023
124 Ali 0
341 Ayesha 27
```

Java Copy Constructor

There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in Java. They are:

- By constructor
- By assigning the values of one object to another
- By clone() method of Object class

Example: By copying values of one object into another with constructor:

```
1
2  public class student {
3       int id;
4       String name;
5       int age;
6  //creating a parameterized constructor  (2 arguments)
7      student(int i,String n){
8      id = i;
9      name = n;
10     }
11 //creating a parameterized constructor  (3 arguments)
12     student(student s){
13     id = s.id;
14     name = s.name;
15     }
16
17     //method to display the values
18     void display()
19     {
20         System.out.println(id+" "+name);
21         }
22
23     public static void main(String args[]){
24     //creating objects and passing values
25     student s1 = new student(124,"Ali");
26     student s2 = new student(s1);
27     //calling method to display the values of object
28     s1.display();
29     s2.display();
30    }
```

Problems Javadoc Declaration Console ×

<terminated> student [Java Application] C:\Users\larai\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v202308

```
124 Ali
124 Ali
```

Example: Without Using Constructor:

```
1
2  public class student {
3       int id;
4       String name;
5  //creating a parameterized constructor  (2 arguments)
6      student(int i,String n){
7      id = i;
8      name = n;
9      }
10 //creating a parameterized constructor  (3 arguments)
11    student() {}
12
13     //method to display the values
14     void display()
15     {
16         System.out.println(id+" "+name);
17         }
18
19     public static void main(String args[]){
20     //creating objects and passing values
21     student s1 = new student(124,"Ali");
22     student s2 = new student();
23     //copying values of object 1 into another
24     s2.id=s1.id;
25     s2.name=s1.name;
26     //calling method to display the values of object
27     s1.display();
28     s2.display();
29    }
```

Problems Javadoc Declaration Console ×

<terminated> student [Java Application] C:\Users\larai\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v20230801-205

```
124 Ali
124 Ali
```

TASKS:

1- Write a JAVA program that creates a default constructor. (Example1)
2- Write a JAVA program that creates 3 parameterized constructors, each with different parameters, and create objects accordingly. (Example 4)
3- Write a JAVA program to create parameterized constructors using 'this' reference.
4- Write a JAVA program to copy the values of one object into another with a constructor (Example 5) and without a constructor (Example 6)

Lab 07

Topics to be covered:

- Object Passing in JAVA
- Equal methods in JAVA
- Copy Objects in JAVA (shallow and deep copy)

Object Passing in JAVA:

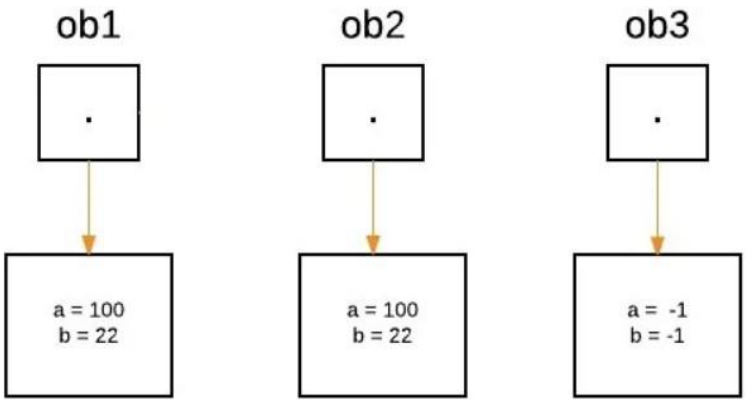
In Java, the way values are passed differs depending on whether it is a primitive type or a reference type. When a primitive type is passed to a method, it is passed by value. However, when an object is passed, it becomes a hybrid of pass-by-value and pass-by-reference. This means that the function cannot directly change the object parameter, but the function can call a method within the parameter to change itself.

- When we create a variable of a class type, we merely create a pointer to an object. Hence, if we pass this pointer to a method, the parameter that takes it will point to the same object as the one pointed to by the argument.
- This implies that methods receive objects as if they were passed through call-by-reference.
- The modifications made to the object within the method are mirrored in the object that is passed as an argument**.**

Example:

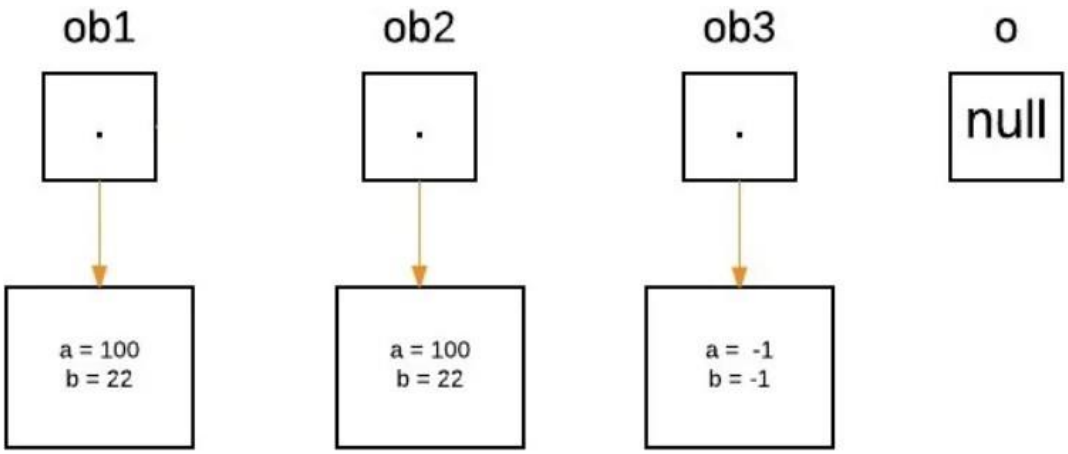
Consider the following scenario where three objects are created as follows:

```
ObjectPassDemo ob1 = new ObjectPassDemo(100, 22);
ObjectPassDemo ob2 = new ObjectPassDemo(100, 22);
ObjectPassDemo ob3 = new ObjectPassDemo(-1, -1);
```



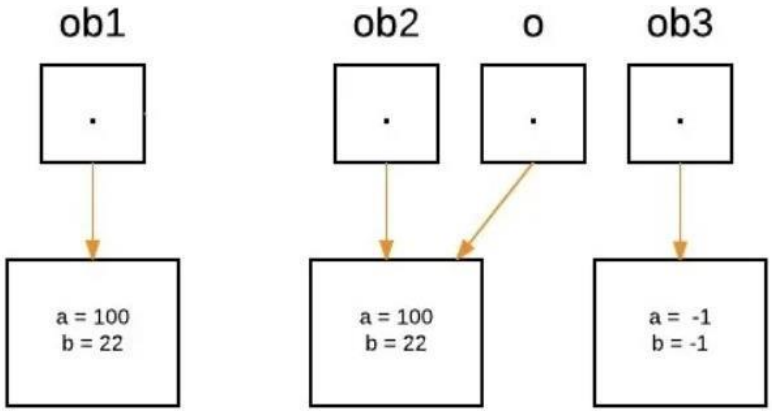
From the method side, a reference of type Foo is declared and initialized as NULL

```
boolean equalTo(ObjectPassDemo o);
```



Upon invoking the equalTo method, the object passed as an argument will be assigned to the reference 'o'. Therefore, in the following statement, 'o' will refer to 'ob2'.

```
System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));
```

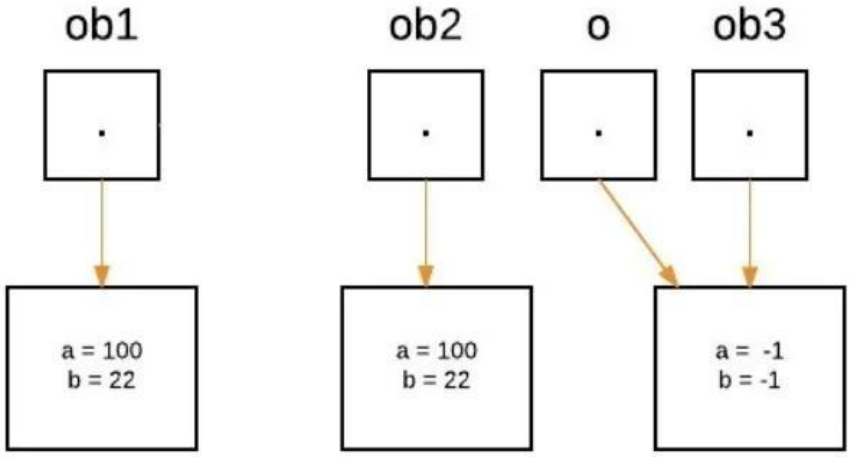


The 'equalTo' method is invoked on 'ob1', while 'o' refers to 'ob2'. If the values of 'a' and 'b' are equivalent for both references, the 'if' condition evaluates to true, and a boolean true is returned.

```
if(o.a == a && o.b == b)
```

Once the following statement is executed, the variable 'o' will be reassigned to 'ob3' again.

```
System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));
```



The method equalTo is invoked on 'ob1' while 'o' points to 'ob3'. Since the variables 'a' and 'b' have different values for both references, the else block will be executed and false will be returned if the condition is false.

```
// Java Program to Demonstrate Objects Passing to Methods.
// Helper class
class ObjectPassDemo {
        int a, b;
        // Constructor
        ObjectPassDemo(int i, int j)
        {
                a = i;
                b = j;
        }
// Method
        boolean equalTo(ObjectPassDemo o)
        {
                // Returns true if o is equal to the invoking
                // object notice an object is passed as an
                // argument to method
                return (o.a == a && o.b == b);
        }
}
// Main class
public class GFG {
        public static void main(String args[])
        {
            ObjectPassDemo ob1 = new ObjectPassDemo(100, 22);
           ObjectPassDemo ob2 = new ObjectPassDemo(100, 22);
           ObjectPassDemo ob3 = new ObjectPassDemo(-1, -1);
           System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));
           System.out.println("ob1 == ob3: "  + ob1.equalTo(ob3));
                                                        }
                                                }
```

```
// Java program to Demonstrate One Object to initialize another
class Box {
        double width, height, depth;
// Notice this constructor. It takes an object of type Box. This constructor use one object to initialize another
        Box(Box ob)
        {       width = ob.width;
                height = ob.height;
                depth = ob.depth;
        }
// constructor used when all dimensions specified
        Box(double w, double h, double d)
        {       width = w;
                height = h;
                depth = d;
        }
        double volume() { return width * height * depth; }
}
public class GFG {
            public static void main(String args[])
        {     Box mybox = new Box(10, 20, 15);
               // Creating a copy of mybox
              Box myclone = new Box(mybox);
              double vol;
              vol = mybox.volume();   //get volume of mybox
             System.out.println("Volume of mybox is " + vol);
             vol = myclone.volume();   //get volume of myclone
             System.out.println("Volume of myclone is " + vol);
                                                                }
                                                              }
```

Returning Objects:

```
// Java Program to Demonstrate Returning of Objects
class ObjectReturnDemo {
        int a;
        ObjectReturnDemo(int i) { a = i; }          // Constructor
        ObjectReturnDemo incrByTen(){
                ObjectReturnDemo temp = new ObjectReturnDemo(a + 10);
                return temp;}  // Method returns an object

}

public class GFG {
        public static void main(String args[])
        {
                ObjectReturnDemo ob1 = new ObjectReturnDemo(2);
                ObjectReturnDemo ob2;

                ob2 = ob1.incrByTen();

                System.out.println("ob1.a: " + ob1.a);
                System.out.println("ob2.a: " + ob2.a);
        }
}
```

Copying Objects in JAVA: Shallow copy & Deep copy

- **Shallow Copy**

  When we do a copy of some entity to create two or more entities such that changes in one entity are reflected in the other entities as well, then we can say we have done a shallow copy. In shallow copy, new memory allocation never happens for the other entities, and the only reference is copied to the other entities. The following example demonstrates the same.

```
class ABC
{
// instance variable of the class ABC
int x = 30;
}
public class ShallowCopyExample
{
// main method
public static void main(String argvs[])
{
// creating an object of the class ABC
ABC obj1 = new ABC();

// it will copy the reference, not value
ABC obj2 = obj1;

// updating the value to 6
// using the reference variable obj2
obj2.x = 6;

// printing the value of x using reference variable obj1
System.out.println("The value of x is: " + obj1.x);
}
}
```

- **Deep Copy**

When we do a copy of some entity to create two or more entities such that changes in one entity are not reflected in the other entities, then we can say we have done a deep copy. In the deep copy, a new memory allocation happens for the other entities, and the reference is not copied to the other entities. Each entity has its independent reference. The following example demonstrates the same.

```
class ABC
{
// instance variable of the class ABC
int x = 30;
}
public class DeepCopyExample
{
// main method
public static void main(String argvs[])
{
// creating an object of the class ABC
ABC obj1 = new ABC();

// it will copy the reference, not value
ABC obj2 = new ABC();

// updating the value to 6
// using the reference variable obj2
obj2.x = 6;

// printing the value of x using reference variable obj1
System.out.println("The value of x is: " + obj1.x);
}
}
```

Copy Constructor in JAVA

```
class GPA{
int firstYear;
int secondYear;
GPA(int fy, int sy){
this.firstYear = fy;
this.secondYear = sy;}
GPA(GPA g){
this.firstYear = g.firstYear;
this.secondYear = g.secondYear;}
public int getFirstYear() {
return firstYear;}
public void setFirstYear(int firstYear) {
this.firstYear = firstYear;}
public int getSecondYear() {
return secondYear;}
public void setSecondYear(int secondYear) {
this.secondYear = secondYear;
}}
class Student{
private String name;
private GPA gpa;
Student(String name, GPA gpa){
this.name = name;
this.gpa = gpa;}
Student(Student s){
this.name = new String(s.name);
this.gpa = new GPA(s.gpa);}
```

```
public String getName(){
        return this.name;
}
public GPA getGPA(){
       return this.gpa;
}
public void setName(String name){
        this.name = name;}
public void setGPA(GPA g){
        this.gpa = g;
}}
public class CopyDemo{
public static void main(String[] args) {
     GPA g = new GPA(7, 8);
     Student s = new Student("Justin", g);  //Original Object
     Student copy = new Student(s);     //Deep copy
     System.out.println("Original Object's GPA: " + s.getGPA().getFirstYear() + "s.getGPA().getSecondYear());
    System.out.println("Cloned Object's GPA: " + copy.getGPA().getFirstYear() + " " +copy.getGPA().getSecondYear());

    copy.getGPA().setFirstYear(10);  //Changing the GPA field of the deep copy
    System.out.println("\nAfter changing the Deep copy");
    System.out.println("Original Object's GPA: " + s.getGPA().getFirstYear() + " " + s.getGPA().getSecondYear());
System.out.println("Cloned Object's GPA: " + copy.getGPA().getFirstYear() + " " + copy.getGPA().getSecondYear());
           }
}
```

Equals Method in JAVA:

- **==**
- **.Equals()**

Both the equals() method and the == operator are used to compare two objects in Java. == is an operator and equals() is method. But == operator compares the reference or memory location of objects in a heap, whether they point to the same location or not.

Whenever we create an object using the operator new, it will create a new memory location for that object. So we use the == operator to check memory location or address of two objects are the same or not. In general, both equals() and "==" operators in Java are used to compare objects to check equality, but here are some of the differences between the two:

- The main difference between the .equals() method and == operator is that one is a method, and the other is the operator.
- We can use == operators for reference comparison (address comparison) and .equals() method for content comparison. In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.

```
// Java program to understand  the concept of == operator
public class Test {
public static void main(String[] args){
        String s1 = "HELLO";
        String s2 = "HELLO";
        String s3 = new String("HELLO");
       System.out.println(s1 == s2); // true
       System.out.println(s1 == s3); // false
       System.out.println(s1.equals(s2)); // true
       System.out.println(s1.equals(s3)); // true
                                                    }
                                                }
```

```
public class Test {
   public static void main(String[] args)
   {
      Thread t1 = new Thread();
      Thread t2 = new Thread();
      Thread t3 = t1;

      String s1 = new String("GEEKS");
      String s2 = new String("GEEKS");

      System.out.println(t1  ==  t3);
      System.out.println(t1  ==  t2);
      System.out.println(s1 == s2);

     System.out.println(t1.equals(t2));
      System.out.println(s1.equals(s2));
   }
}
```

Lab 08

Revision on Classes

Create a class Account that has the following instance variables:

a. private account number (string)
b. private account holder name (string)
c. private balance (floating-point)
d. private account opening date (string)
e. profitRate (final, floating-point): 5% for all accounts

Supply a default constructor, a parameterized constructor, and a copy constructor.

Write a setter and a getter for each private instance variable.

Also, include the following methods:

a. withdraw: deducts a specified amount from the balance
b. deposit: adds a specified amount to the balance
c. accountStatement: displays all information of an account
d. creditProfit: calculates the profit using profitRate and updates account balance

Write another class Bank, which acts as your main class. Inside the main method, your program shall provide a console-based menu to the user with the following options:

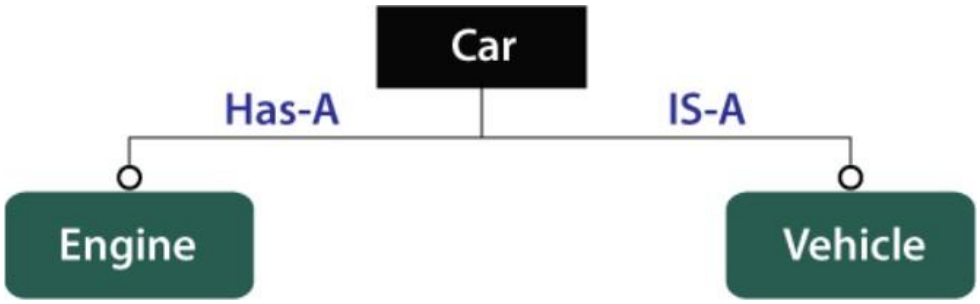
1. Withdraw an Amount (required input: account number, amount to withdraw)
2. Deposit an Amount (required input: account number, amount to deposit)
3. Account Statement (required input: account number)
4. Credit Profit (required input: account number)
5. Exit Program

Lab No 09

Composition in JAVA

The Composition is a way to design or implement the **"has-a"** relationship. The "has-a" relationship is used to ensure the code reusability in our program. In Composition, we use an instance variable that refers to another object.

The composition relationship of two objects is possible when one object contains another object, and that object is fully dependent on it. The contained object should not exist without the existence of its parent object. Simply, it is a technique to describe the reference between two or more classes. And for that, we use the instance variable, which should be created before it is used.



- The Composition represents a part-of relationship.
- Both entities are related to each other in the Composition.
- The Composition between two entities is done when an object contains a composed object, and the composed object cannot exist without another entity.

For example, if a university HAS-A college-lists, then a college is a whole, and college-lists are parts of that university.

- If a university is deleted, then all corresponding colleges for that university should be deleted.

Lab Task 01: We create a class **College** that contains variables, i.e., name and address. We also create a class **University** that has a reference to refer to the list of colleges. A University can have more than one collages. So, if a university is permanently closed, then all colleges within that particular university will be closed because colleges cannot exist without a university. The relationship between the university and colleges is Composition.

```
import java.io.*;
import java.util.*;
// class College
class College {
    public String name;
    public String address;
    College(String name, String address)
    {
        this.name = name;
         this.address = address;
         }    }
// University has more than one college.
class University {
    // reference to refer to list of college.
    private final List<College> colleges;
    University(List<College> colleges)
    {
        this.colleges = colleges;
    }
    // Getting total number of colleges
         public List<College> getTotalCollegesInUniversity()
         {
             return colleges;
         }
     }
```

```
class CompositionExample {
    public static void main(String[] args)
    {
        // Creating the Objects of College class.
        College c1   = new College("ABES Engineering College", "Ghaziabad");
        College c2   = new College("AKG Engineering College", "Ghaziabad");
        College c3    = new College("ACN College of Engineering & Management Sudies",    "Alig
arh");
    // Creating list which contains the no. of colleges.
        List<College> college = new ArrayList<College>();
        college.add(c1);
        college.add(c2);
        college.add(c3);
        University university = new University(college);
        List<College> colleges = university.getTotalCollegesInUniversity();
        for (College cg : colleges) {
            System.out.println("Name : " + cg.name  + " and "    + " Address : " + cg.address);
        }
    }
}
```

Lab Task 02: Library system

Let's understand the composition in Java with the example of books and library. In this example, we create a class Book that contains data members like author, and title and create another class Library that has a reference to refer to the list of books. A library can have no. of books on the same or different subjects. So, if the Library gets destroyed then all books within that particular library will be destroyed. i.e., books cannot exist without a library. The relationship between the library and books is composition.

```
// Java program to Illustrate Concept of Composition

// Importing required classes
import java.io.*;
import java.util.*;

// Class 1
// Helper class
// Book class
class Book {

    // Member variables of this class
    public String title;
    public String author;

    // Constructor of this class
    Book(String title, String author)
    {

        // This keyword refers top current instance
        this.title = title;
        this.author = author;
    }
}
```

```
// Class 2
// Helper class
// Library class contains list of books.
class Library {

    // Reference to refer to list of books.
    private final List<Book> books;

    // Constructor of this class
    Library(List<Book> books)
    {

        // This keyword refers to current instance itself
        this.books = books;
    }

    // Method of this class
    // Getting the list of books
    public List<Book> getListOfBooksInLibrary()
    {
        return books;
    }
}

// Class 3
// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Creating the objects of class 1 (Book class)
        // inside main() method
```

```
        Book b1  = new Book("EffectiveJ Java", "Joshua Bloch");
        Book b2  = new Book("Thinking in Java", "Bruce Eckel");
        Book b3 = new Book("Java: The Complete Reference", "Herbert Schildt");

        // Creating the list which contains the
        // no. of books.
        List<Book> book = new ArrayList<Book>();

        // Adding books to List object
        // using standard add() method
        book.add(b1);
        book.add(b2);
        book.add(b3);

        // Creating an object of class 2
        Library library = new Library(book);

        // Calling method of class 2 and storing list of
        // books in List Here List is declared of type
        // Books(user-defined)
        List<Book> books = library.getListOfBooksInLibrary();

        // Iterating over for each loop
        for (Book bk : books) {

            // Print and display the title and author of
            // books inside List object
            System.out.println("Title : " + bk.title   + " and "  + " Author : " + bk.author);
        }
    }
}
```

Lab Task 03: Engine and Car Class

```
public class Engine
{
  private String type;
  private int horsePower;

  Engine(String type, int horsePower) {
    this.type = type;
    this.horsePower = horsePower;
  }
  public String getType() {
    return type;
  }
  public int getHorsePower() {
    return horsePower;
  }
  public void setType(String type){
    this.type = type;
  }
  public void setHorsePower(int horsePower){
    this.horsePower = horsePower;
  }
}

public class Car
{
  private final String name;
  private final Engine engine; // Composition.

  public Car(String name, Engine engine) {
    this.name = name;
    this.engine = engine;
  }
  public String getName() {
    return name;
  }
  public Engine getEngine() {
    return engine;
```

```
  }
}

public class Test {
public static void main(String[] args)
{
// Creating an object of Engine class.
  Engine engn = new Engine("Petrol", 300);

// Creating an object of Car class.
  Car car = new Car("Alto", engn);
  System.out.println("Name of car: " +car.getName()+ "\n" +"Type of engine: "
+engn.getType()+ "\n" + "Horse power of Engine: " +engn.getHorsePower());
 }
}
```

Submission Instructions:

Submit the code and output screenshots of the above tasks.

IQRA UNIVERSITY IU

Object Oriented Programming

Lab# 10 Tasks

Note: Note: Submit code execution screenshots with their output Results (Create Folder with your name)

1. Lab Task 1: Implementation of inheritance

```
public class A {
   int i, j;
   void showij() {
      System.out.println("i and j: " + i + " " + j);
   }
}
class B extends A {
   int k;
   void showk() {
      System.out.println("k: " + k);
   }
   void sum() {
      System.out.println("i+j+k: " + (i + j + k));
   }
}
class SimpleInheritance {
   public static void main(String args[]) {
      A superOb = new A();
      B subOb = new B();
      superOb.i = 10;
      superOb.j = 20;
      System.out.println("Contents of superOb: ");
      superOb.showij();
      System.out.println();
      subOb.i = 7;
      subOb.j = 8;
      subOb.k = 9;
      System.out.println("Contents of subOb: ");
      subOb.showij();
      subOb.showk();
      System.out.println();
      System.out.println("Sum of i, j and k in subOb:");
      subOb.sum();
```

```
        }
    }
```

2. Lab Task 2 Implementation of Inheritance

```
class Box {
    double width;
    double height;
    double depth;
    // construct clone of an object
    Box(Box ob) { // pass object to constructor
        width = ob.width;
        height = ob.height;
        depth = ob.depth;
    }
    // constructor used when all dimensions specified
    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }
    // constructor used when no dimensions specified
    Box() {
        width = -1; // use -1 to indicate
        height = -1; // an uninitialized
        depth = -1; // box
    }
    // constructor used when cube is created
```

```
    Box(double len) {
        width = height = depth = len;
    }
    // compute and return volume
    double volume() {
        return width * height * depth;
    }
}
// Here, Box is extened to include weight.
class BoxWeight extends Box {
    double weight; // weight of box
    // constructor for BoxWeight
    BoxWeight(double w, double h, double d, double m) {
        width = w;
        height = h;
        depth = d;
        weight = m;
    }
}
class DemoBoxWeight {
    public static void main(String args[]) {
        BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
        BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
        double vol;
        vol = mybox1.volume();
```

```
        System.out.println("Volume of mybox1 is " + vol);
        System.out.println("Weight of mybox1 is " + mybox1.weight);
        System.out.println();
        vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);
        System.out.println("Weight of mybox2 is " + mybox2.weight);
    }
}
```

IQRA UNIVERSITY IU

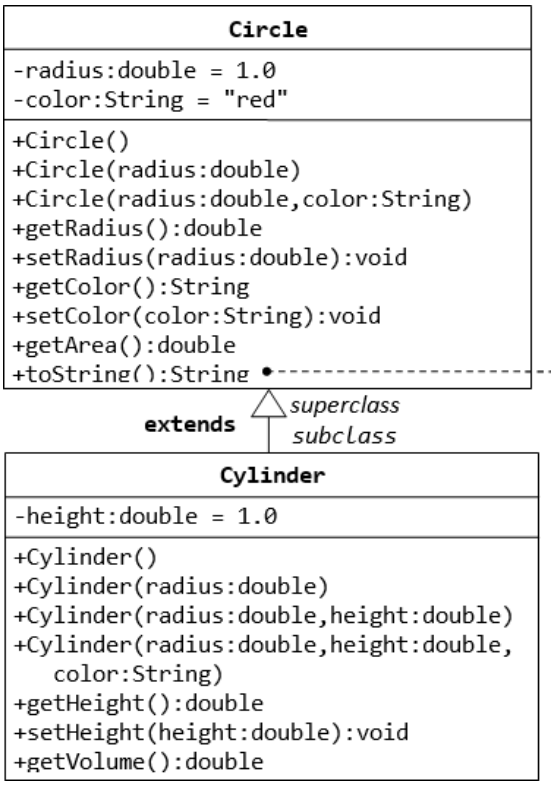
Object Oriented Programming

Lab# 11 Tasks

Note: Note: Submit code execution screenshots with their output Results (Create Folder with your name)

1. Lab Task **01 (Implementing the Circle and Cylinder classess)**

Implement the concept of ***Inheritance*** as per the details below:



The subclass `Cylinder` should invokes the **superclass**' constructors (via `super`) and inherits the variables and methods from the **superclass** `Circle`.

- Write a test program (says `TestCylinder`) to test the `Cylinder` class by creating 3 objects to print radius, height, base area and volume of 3 Cylinders having different radius.

IQRA UNIVERSITY IU

```java
public class Circle {
    private double radius = 1.0;
    private String color = "red";

    // default Constructor
    Circle() {
        radius = 2.5;
        color = "Green";
    }

    // single Parameter constructor
    Circle(double radius) {
        this.radius = radius;
    }

    // two Parameter constructor
    Circle(double radius, String color) {
        this.radius = radius;
        this.color = color;
    }

    // getter setter for access private variables
    void setRadius(double r) {
        radius = r;
    }

    double getRadius() {
        return radius;
    }

    void setColor(String c) {
        color = c;
    }

    String getColor() {
        return color;
    }

    // area of circle πr2
    double getArea() {
        return 3.14 * radius * radius;
    }

    public String toString() {
```

```
        return "The Radius of the circle is " + this.getRadius() + ". The color
of the circle is " + this.getColor();
    }
}
```

```
public class Cylinder extends Circle {
    double height = 1.0;

    Cylinder() {
        this.height = 2.5;
    }

    Cylinder(double radius) {
        super(radius);
    }

    Cylinder(double radius, double height) {
        super(radius);
        this.height = height;
    }

    Cylinder(double radius, double height, String color) {
        super(radius, color);
        this.height = height;
    }

    void setHeight(double height) {
        this.height = height;
    }

    double getHeight() {
        return height;
    }

    double getVolume() {
        return 3.14 * getRadius() * getRadius() * getHeight();
    }
}
```

```
public class CylMain {
    public static void main(String[] args) {
```

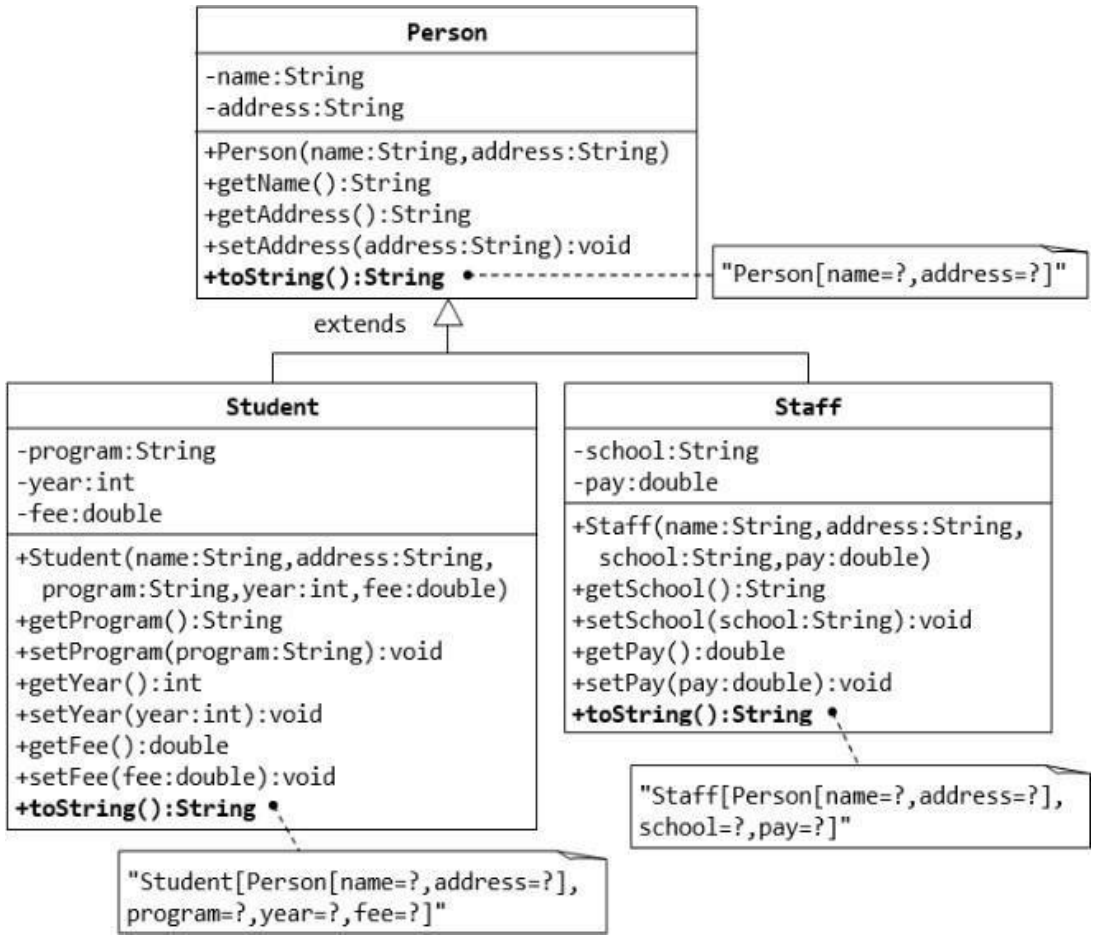
```
        // pass value to default constructor
        Cylinder c = new Cylinder();
        // pass value to radius constructor
        Cylinder c1 = new Cylinder(2.9);
        // pass value to radius and height constructor
        Cylinder c2 = new Cylinder(5.3, 5.8);
        // pass value to radius, height and color constructor
        Cylinder c3 = new Cylinder(1.2, 4.5, "Orange");
        System.out.println(" Area of Circle : " + c.getArea());
        System.out.println(" Volume of Cylinder : " + c.getVolume());
        System.out.println(" Area of Circle : " + c1.getArea());
        System.out.println(" Volume of Cylinder : " + c1.getVolume());
        System.out.println(" Area of Circle : " + c2.getArea());
        System.out.println(" Volume of Cylinder : " + c2.getVolume());
        System.out.println(" Area of Circle : " + c3.getArea());
        System.out.println(" Volume of Cylinder : " + c3.getVolume());
        System.out.println(c.toString());
        System.out.println(c1.toString());
        System.out.println(c2.toString());
        System.out.println(c3.toString());

    }
}
```

Output

```
PROBLEMS 89   OUTPUT   DEBUG CONSOLE   TERMINAL

:\Users\SmN\AppData\Roaming\Code\User\workspaceStorage\8e4cd5bf2bb9c568d048ea23ff178963\redhat.java\jdt_ws\Java_72da403b\bin' 'C
ylMain'
 Area of Circle : 19.625
 Volume of Cylinder : 49.0625
 Area of Circle : 26.4074
 Volume of Cylinder : 26.4074
 Area of Circle : 88.20259999999999
 Volume of Cylinder : 511.5750799999999
 Area of Circle : 4.521599999999999
 Volume of Cylinder : 20.347199999999997
The Radius of the circle is 2.5. The color of the circle is Green
The Radius of the circle is 2.9. The color of the circle is red
The Radius of the circle is 5.3. The color of the circle is red
The Radius of the circle is 1.2. The color of the circle is Orange
PS D:\Java>
```

2. Lab Task 02 (Implementing Super class Person and its subclasses)

Implement the following classes as per the details below




```
class OverPerson{
    private String name;
    private String address;

    OverPerson(String n,String a){
        name=n;
        address=a;
    }
    String getName(){
        return this.name;
    }
    void setName(String n){
        name=n;
    }

    String getAddress(){
        return this.address;
    }
```

IQRA UNIVERSITY IU

```
    void setAddress(String a){
        address=a;
    }
    public String toString(){
        return "Person : "+ name+"  Address : "+address;
    }
}
class Student extends OverPerson{
private String program;
private int year;
private double fee;

Student (String n,String a,String p,int y,double f){
    super(n,a);
    program=p;
    year=y;
    fee=f;
}
String getProgram(){
    return this.program;
}
void setProgram(String p){
    program=p;
}
    int  getYear(){
        return this.year;
    }
    void setYear(int y){
        year=y;
    }
    double getfee(){
        return this.fee;
    }
    void setFee(double f){
        fee=f;
    }
public String toString(){
    return "Person : "+ super.getName()+"  Address :
"+super.getAddress()+"Program : "+program+"  Year : "+year+"  fee : "+fee;
}
}
class Staff extends OverPerson{
    private String school;
    private double pay;
```

```
    Staff(String n,String a,String s,double p){
        super(n,a);
        school=s;
        pay=p;
    }
    String getSchool(){
        return this.school;
    }
    void setSchool(String s){
        school=s;
    }
    double getPay(){
        return this.pay;
    }
    void setFee(double p){
        pay=p;
    }
    public String toString(){
        return "Person : "+ super.getName()+"  Address :
"+super.getAddress()+"School : "+school+"  Pay : "+pay;
    }
}



import java.util.Scanner;

public class OverMain {
    public static void main(String[] args) {

        Person  obj2  =  new  Person("Junaid",  "H#12,St#26,f-6,Islamabad");
        Student obj3 = new Student(obj2.getName(), obj2.getAddress(), "BSAI",
2021, 88000);
        Staff obj4 = new Staff("Ms.Sara", "H#41,St#14,f-11,Islamabad",
"CaliforniaUniversity,US", 100000);
        System.out.println(obj2);
        System.out.println(obj3);
        System.out.println(obj4);
        Scanner obj = new Scanner(System.in);
        System.out.println("\n\nStudent, Please enter the
name,address,program,year and fee");
        String name = obj.next();
        String address = obj.next();
        String program = obj.next();
        int year = obj.nextInt();
```

```
        double fee = obj.nextDouble();
        obj3 = new Student(name, address, program, year, fee);
        System.out.println(obj3);
        obj.close();
    }

}
```

3. Lab Task 03 (Implementing Super class Shape and its subclasses)

Write a superclass called ***Shape*** which contains:

- Two instance variables color (String) and filled (boolean).
- Two constructors: a no-arg (no-argument) constructor that initializes the color to "green" and filled to true, and a constructor that initializes the color and filled to the given values.
- Getter and setter for all the instance variables.
- A toString() method that returns "A Shape with color of xxx and filled/Not filled".

Write a test program to test all the methods defined in Shape.

Write two subclasses of Shape called ***Circle*** and ***Rectangle***, as shown in the class diagram.
The Circle class contains:

- An instance variable **radius** (double).
- Three constructors The no-arg constructor initializes the radius to 1.0.
- Getter and setter for the instance variable radius.
- Methods **getArea()** and **getPerimeter().**
- Override **the toString()** method inherited, to return "A Circle with radius=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

The ***Rectangle*** class contains:

- Two instance variables **width** (double) and **length** (double).
- Three constructors as shown. The no-arg constructor initializes the width and length to 1.0.
- Getter and setter for all the instance variables.
- Methods **getArea()** and **getPerimeter().**
- Override the **toString()** method inherited, to return "A Rectangle with width=xxx and length=zzz, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

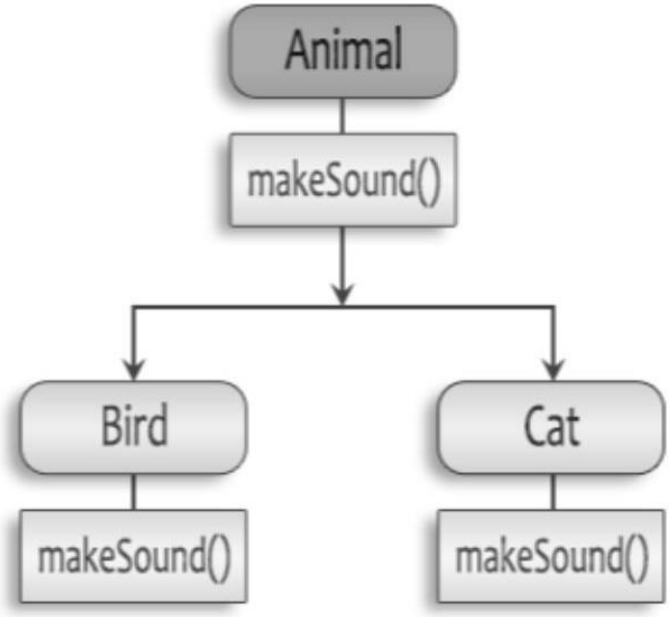
Write a class called ***Square***, as a subclass of Rectangle. Convince yourself that Square can be modeled as a subclass of Rectangle. Square has no instance variable, but inherits the instance variables **width** and **length** from its superclass Rectangle.

Lab 12

Run-time Polymorphism

Method Overriding

Lab Task 1: Write a Java program to create a base class Animal (Animal Family) with a method called Sound(). Create two subclasses Bird and Cat. Override the Sound() method in each subclass to make a specific sound for each animal.

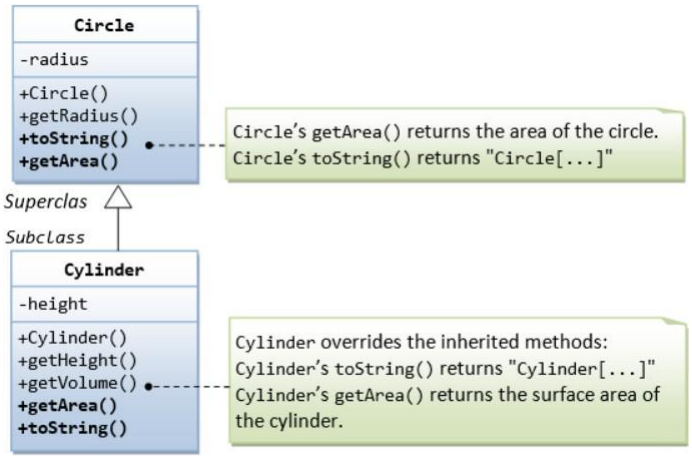



```
class Animal{
   void sound()
   {
      System.out.println("Animal makes a sound");
```

```
    }
}

class Bird extends Animal{
    @Override
    void sound(){
        System.out.println("Birds make the sound of Chirp");
    }
}

class Cat extends Animal{
    @Override
    void sound(){
        System.out.println("Cats make the sound of Meow Meow!");
    }
}

public class Main{
    public static void main (String[]args){
        Animal obj = new Animal();
        obj.sound();
        Bird obj1 = new Bird();
        obj1.sound();
        Cat obj2 = new Cat();
        obj2.sound();
```

```
    }
}
```

Lab Task 2: Implement the following:




```
class Circle{
    private double radius;

    public Circle(double r){
        this.radius = r;
    }

    public double getradius(){
        return this.radius;
    }

    public double getArea(){
```

```
        return 3.14*this.radius*this.radius;
    }

    public String toString(){
        return "The radius of the circle is :" +this.radius ;
    }
}

class Cylinder extends Circle{
    private double height;
    public Cylinder(double radius, double h){
        super(radius);
        this.height = h;
    }

    public double getHeight(){
        return this.height;
    }

    public double getVolume(){
        return super.getArea()*height;
    }
    @Override
    public double getArea()
    {
```

```
        return 2*3.14*getradius()*height;
    }
    @Override
    public String toString(){
        return "The height of cylinder is :"+this.height+ "," +super.toString();
    }


}


public class Main{
    public static void main (String[]args){
        Circle c = new Cylinder(1.1,1.3);
        c.getradius();
        c.toString();
        c.getArea();
    }
}
```

*we can create an instance of `Cylinder`, and assign it to a `Circle` (its superclass) reference, You can invoke all the methods defined in the `Circle` class for the reference `c`, (which is actually holding a `Cylinder` object). This is because a subclass instance possesses all the properties of its superclass.

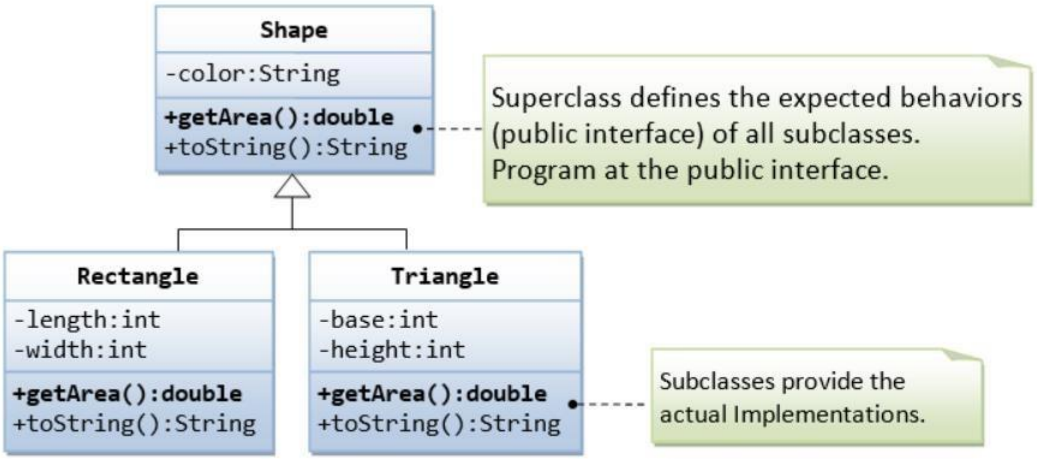
However, you CANNOT invoke methods defined in the `Cylinder` class for the reference `c.`

This is because `c` is a reference to the `Circle` class, which does not know about methods defined in the subclass `Cylinder`.

`c` is a reference to the `Circle` class, but holds an object of its subclass `Cylinder`. The reference `c`, however, *retains its internal identity*. In our example, the subclass `Cylinder` overrides methods `getArea()` and `toString()`. `c.getArea()` or `c.toString()` invokes the *overridden* version defined in the subclass `Cylinder`, instead of the version defined in `Circle`. This is because `c` is in fact holding a `Cylinder` object internally. *

- A subclass instance can be assigned (substituted) to a superclass' reference.
- Once substituted, we can invoke methods defined in the superclass; we cannot invoke methods defined in the subclass.
- However, if the subclass overrides inherited methods from the superclass, the subclass (overridden) versions will be invoked.

Lab Task 3: Implement the following



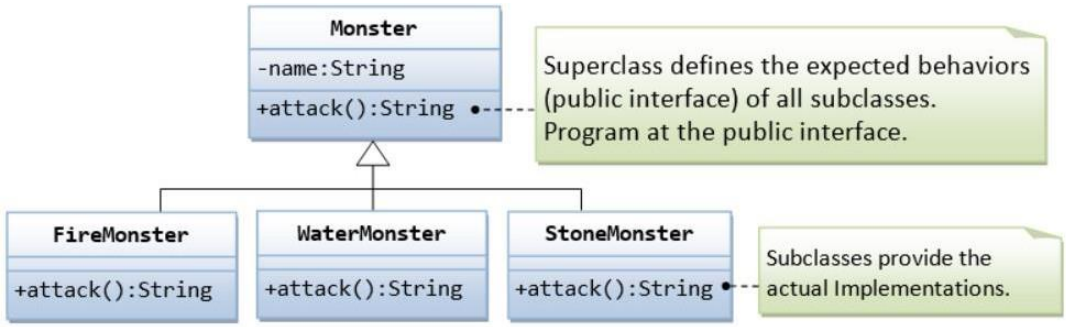

```
class Shape {

  private String color;
  public Shape (String color) {
    this.color = color;
  }
```

```
  @Override
  public String toString() {
    return "Shape[color=" + color + "]";
  }

  public double getArea() {
    System.err.println("Shape unknown! Cannot compute area!");
    return 0;
  }
}
class Rectangle extends Shape {
  private int length, width;
  public Rectangle(String color, int length, int width) {
    super(color);
    this.length = length;
    this.width = width;
  }

  @Override
  public String toString() {
    return "Rectangle[length=" + length + ",width=" + width + "," +
super.toString() + "]";
  }
  @Override
  public double getArea() {
```

```
    return length*width;
  }
}
class Triangle extends Shape {
  private int base, height;
  public Triangle(String color, int base, int height) {
    super(color);
    this.base = base;
    this.height = height;
  }

  @Override
  public String toString() {
    return "Triangle[base=" + base + ",height=" + height + "," + super.toString() +
"]";
  }

  @Override
  public double getArea() {
    return 0.5*base*height;
  }
}
public class Main {
  public static void main(String[] args) {
```

```
        Shape s3 = new Shape("green");
        System.out.println(s3);

        System.out.println("Area is " + s3.getArea());

        Shape s1 = new Rectangle("red", 4, 5);
        System.out.println(s1);

        System.out.println("Area is " + s1.getArea());

        Shape s2 = new Triangle("blue", 4, 5);
        System.out.println(s2);

        System.out.println("Area is " + s2.getArea());
    }

}
```

Lab Task 04: Implement the following:



```
class Monster {
  private String name;
  public Monster(String name) {
    this.name = name;
  }

  public String attack() {
    return "!^_&^$@+%$* I don't know how to attack!";
  }
}

class FireMonster extends Monster {
  public FireMonster(String name) {
    super(name);
  }

  @Override
  public String attack() {
    return "Attack with fire!";
  }
}

class WaterMonster extends Monster {
  public WaterMonster(String name) {
    super(name);
```

```
    }

    @Override
    public String attack() {
        return "Attack with water!";
    }
}

class StoneMonster extends Monster {
    public StoneMonster(String name) {
        super(name);
    }
    @Override
    public String attack() {
        return "Attack with stones!";
    }
}

public class Main {
    public static void main(String[] args) {
        Monster m1 = new FireMonster("r2u2");
        Monster m2 = new WaterMonster("u2r2");
        Monster m3 = new StoneMonster("r2r2");

        System.out.println(m1.attack());
```

```
        System.out.println(m2.attack());
        System.out.println(m3.attack());


        m1 = new StoneMonster("a2b2");
        System.out.println(m1.attack());


        Monster m4 = new Monster("u2u2");
        System.out.println(m4.attack());
    }
}
```

IQRA UNIVERSITY IU

Object Oriented Programming

Lab# 13 Tasks

Note: Note: Submit code execution screenshots with their output Results (Create Folder with your name)

1. **Lab Task 1: Implementation Abstract Class**

```
//abstract class
public abstract class AbPerson {
    private String name;
    private String gender;

    public AbPerson(String nm, String gen){
        this.name=nm;
        this.gender=gen;
    }

    //abstract method
    public abstract void work();

    @Override
    public String toString(){
        return "Name="+this.name+"::Gender="+this.gender;
    }

    public void changeName(String newName) {
        this.name = newName;
    }
}
public class Employee extends AbPerson {

    private int empId;

    public Employee(String nm, String gen, int id) {
        super(nm, gen);
        this.empId=id;
    }

    @Override
    public void work() {
```

```
            if(empId == 0){
                System.out.println("Not working");
            }else{
                System.out.println("Working as employee!!");
            }
        }

        public static void main(String args[]){
            //coding in terms of abstract classes
            AbPerson student = new Employee("Dove","Female",0);
            AbPerson employee = new Employee("Pankaj","Male",123);
            student.work();
            employee.work();
            //using method implemented in abstract class - inheritance
            employee.changeName("Pankaj Kumar");
            System.out.println(employee.toString());
        }

    }
```

2. Lab Task 02 (Experimenting for Default Values of Instance Variables)

Implement a class ***Test*** and add 9 different data members in that class for each data type namely (byte, short, int, long, float, double, boolean, char, String). Now in your main class, create an object of the Test class and print the values of data members of that object. Now observe with what values the object will be initialized by default.

```
public class Test {

   private byte aByte;

   private short aShort;

   private int anInt;

   private long aLong;

   private float aFloat;

   private double aDouble;

   private boolean aBoolean;

   private char aChar;

   private String aString;
```

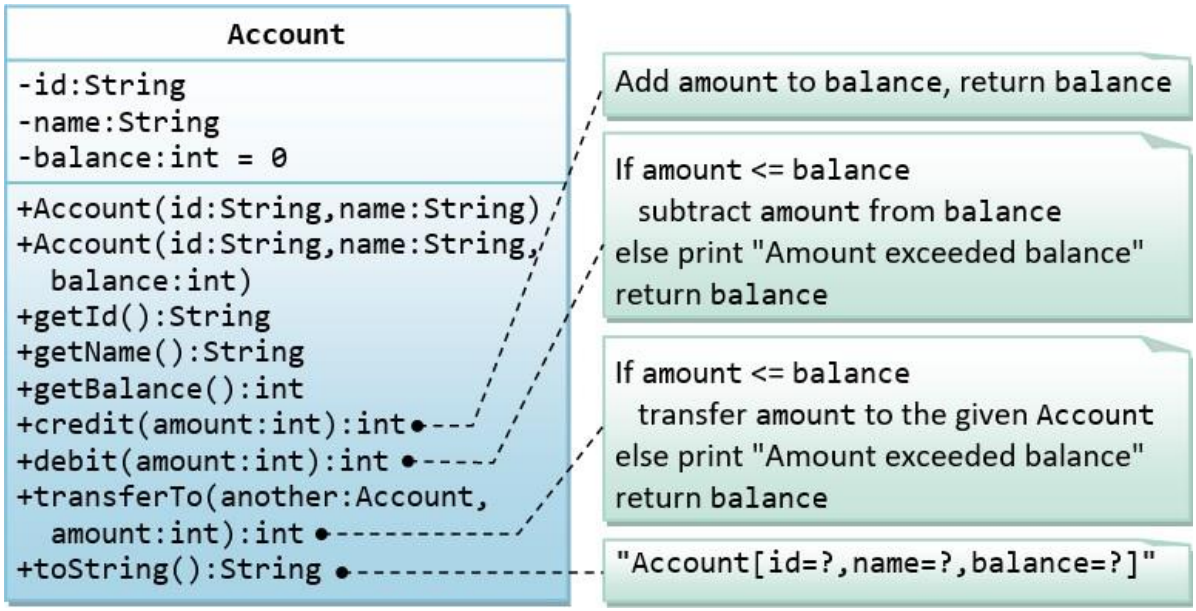
```
    // Constructor
    public Test(byte aByte, short aShort, int anInt, long aLong, float aFloat, double aDouble,
boolean aBoolean, char aChar, String aString) {
        this.aByte = aByte;
        this.aShort = aShort;
        this.anInt = anInt;
        this.aLong = aLong;
        this.aFloat = aFloat;
        this.aDouble = aDouble;
        this.aBoolean = aBoolean;
        this.aChar = aChar;
        this.aString = aString;
    }
    // Getters and setters
    public byte getByte() {
        return aByte;
    }
    public void setByte(byte aByte) {
        this.aByte = aByte;
    }
    public short getShort() {
        return aShort;
    }
public void setShort(short aShort) {
        this.aShort = aShort;
```

```
}
public int getInt() {
    return anInt;
}
public void setByte(int anInt) {
    this.anInt = anInt;
}
public long getLong() {
    return aLong;
}
public void setLong(long aLong) {
    this.aLong = aLong;
}
public float getFloat() {
    return aFloat;
}
public void setFloat(float aFloat) {
    this.aFloat = aFloat;
}
public double getDouble() {
    return aDouble;
}
public void setDouble(double aDouble) {
    this.aDouble = aDouble;
}
```

```
    public boolean getBoolean() {
        return aBoolean;
    }
    public void setBoolean(boolean aBoolean) {
        this.aBoolean = aBoolean;
    }
    public char getChar() {
        return aChar;
    }
    public void setChar(char aChar) {
        this.aChar = aChar;
    }
    public String getString() {
        return aString;
    }
    public void setString(String aString) {
        this.aString = aString;
    }
}

public class Main{
    public static void main(String[]args){
        Test test = new Test((byte)1,(short) 2,3,4,5.0f,6.0,true,'X',"OOP");
        System.out.println(test.getByte());
        System.out.println(test.getFloat());
```

```
        System.out.println(test.getInt());
        System.out.println(test.getDouble());
        System.out.println(test.getChar());
        System.out.println(test.getString());
        System.out.println(test.getBoolean());
        System.out.println(test.getLong());
    }
}
```

3. **Lab Task 3** A class called Account, which models a bank account of a customer, is designed as shown in the following class diagram. The methods credit(amount) and debit(amount) add or subtract the given amount to the balance. The method transferTo(anotherAccount, amount) transfers the given amount from this Account to the given anotherAccount. Write the Account class.

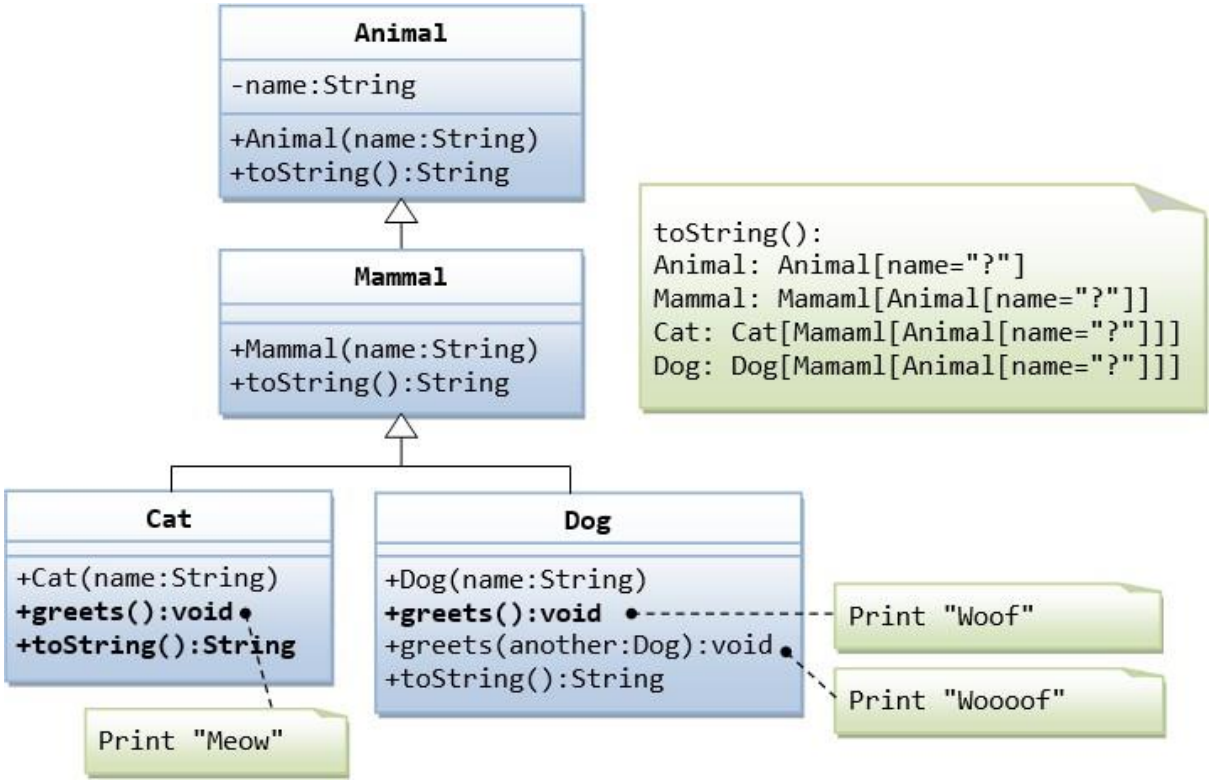



```
class Account{
    private String id;
    private String name;
    private int balance;
```

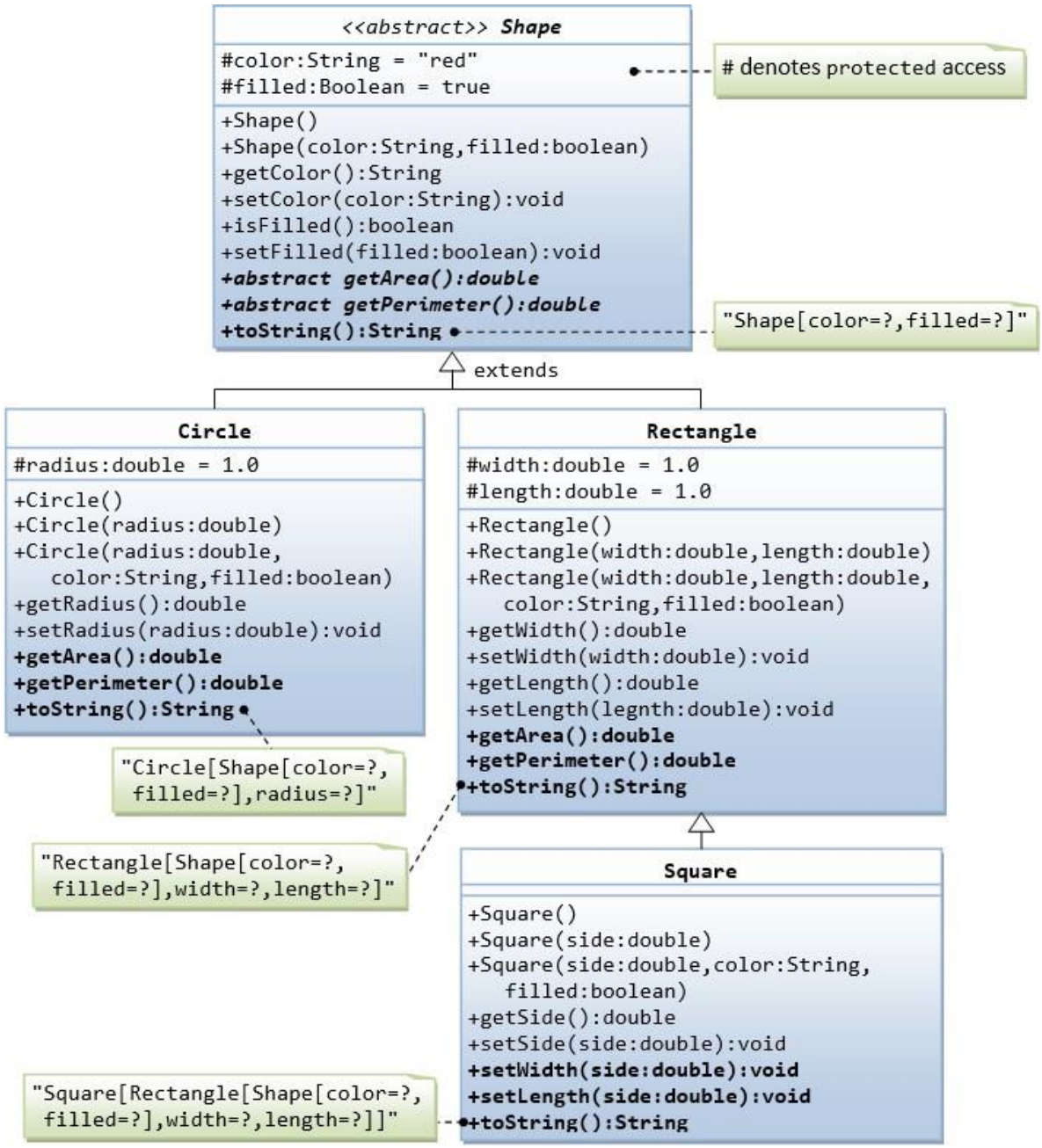
```
public Account(String id, String name){
    this.id = id;
    this.name = name;
    this.balance = 0;
}

public void set_id(String id){
    this.id = id;
}
public String get_id(){
    return this.id;
}
public void set_name(String name){
    this.name = name;
}
public String get_name(){
    return this.name;
}
public int get_bal(){
    return this.balance;
}

public int credit(int amount){
    balance += amount;
    return balance;
}

public int debit(int amount){
    if (amount<=balance){
        balance -=amount;
        System.out.println("Amount Debited");
        return balance;
    }
    else {
        System.out.println("Insufficent Balance");
        return -1;
    }
}

public int transferto(Account anotherAcc, int amount){
```

```
        if (amount<=balance){
            balance -= amount;
            anotherAcc.credit(amount);
            return balance;
        }

        else {
             System.out.println("Insufficent Balance");
            return -1;
        }


    }
    public String toString(){
        return "Id :" + this.id + " Name" + this.name + "balance" + this.balance ;
    }

}


public class Main{
    public static void main(String[]args){
        Account a1 = new Account("234", "Ayesha");
        Account a2 = new Account("235", "Haider");
        a1.credit(10000);
        a1.debit(5000);
        a1.transferto(a2,2500);
        a2.credit(2500);
        System.out.println(a1.toString());
        System.out.println(a2.toString());
    }
    }
```

4. **Task 4: write the codes for all the classes as shown in the class diagram.**



5. Lab task 5: Abstract Superclass Shape and Its Concrete Subclasses
   Shape is an abstract class containing 2 abstract methods: getArea() and getPerimeter(), where its concrete subclasses must provide its implementation. All instance variables shall have protected access, i.e., accessible by its subclasses and classes in the same package



In this exercise, Shape shall be defined as an abstract class, which contains:

- Two protected instance variables color(String) and filled(boolean). The protected variables can be accessed by its subclasses and classes in the same package. They are denoted with a '#' sign in the class diagram.
- Getter and setter for all the instance variables, and toString().
- Two abstract methods getArea() and getPerimeter() (shown in italics in the class diagram).

The subclasses Circle and Rectangle shall *override* the abstract methods getArea() and getPerimeter() and provide the proper implementation. They also *override* the toString().

Object Oriented Programming

Lab# 14 Tasks

Note: Note: Submit code execution screenshots with their output Results (Create Folder with your name)

1. Lab Task 1: Implementation of Interface

```
class Client implements Callback {

    // Implement Callback's interface

    public void callback(int p) {

        System.out.println("callback called with " + p);

    }

    void nonIfaceMeth() {

        System.out.println("Classes that implement interfaces " + "may also define other members, too.");

    }

}

class TestIface {

    public static void main(String args[]) {

        Callback c = new Client();

        c.callback(42);

    }

}
```

```
public interface Callback {

    void callback(int param);

}
```

2. Lab Task 02 (Experimenting for Default Values of Instance Variables)

```
  class AnotherClient implements Callback {
   // Implement Callback's interface
   public void callback(int p) {
      System.out.println("Another version of callback");
      System.out.println("p squared is " + (p * p));
   }
}
class Clients implements Callback {
   // Implement Callback's interface
   public void callback(int p) {
      System.out.println("callback called with " + p);
   }
   void nonIfaceMeth() {
      System.out.println("Classes that implement interfaces " + "may also define other members,
too.");
   }
}
class TestIface2 {
   public static void main(String args[]) {
      Callback c = new Clients();
      AnotherClient ob = new AnotherClient();
      c.callback(42);
      c = ob; // c now refers to AnotherClient object
      c.callback(42);
```

```
  }

}
```

3. **Lab Task 3 Random Number class methods**.

```
import java.util.Random;

public class RandomNumberExample {
   public static void main(String[] args) {
      // initialize random number generator
      Random random = new Random();
      // generates boolean value
      System.out.println(random.nextBoolean());
      // generates double value
      System.out.println(random.nextDouble());
      // generates float value
      System.out.println(random.nextFloat());
      // generates int value
      System.out.println(random.nextInt());
      // generates int value within specific limit
      System.out.println(random.nextInt(20));
   }
}
```

4. **Task 4:**

```
import java.util.Random;
interface SharedConstants {
int NO = 0;
  int YES = 1;
  int MAYBE = 2;
  int LATER = 3;
  int SOON = 4;
  int NEVER = 5;
}

class Question implements SharedConstants {
  Random rand = new Random();

  int ask() {
    int prob = (int) (100 * rand.nextDouble());
    if (prob < 30)
```

```
      return NO;
    else if (prob < 60)
      return YES;
    else if (prob < 75)
    return LATER;
    else if (prob < 98)
    return SOON;
    else
      return NEVER;
  }
}

class AskMe implements SharedConstants {
  static void answer(int result) {
    switch (result) {
      case NO:
        System.out.println("No");
        break;
      case YES:
        System.out.println("Yes");
        break;
      case MAYBE:
        System.out.println("Maybe");
        break;
      case LATER:
        System.out.println("Later");
        break;
      case SOON:
        System.out.println("Soon");
        break;
      case NEVER:
        System.out.println("Never");
        break;
    }
  }

  public static void main(String args[]) {
    Question q = new Question();
    answer(q.ask());
    answer(q.ask());
```

```
        answer(q.ask());
        answer(q.ask());
      }
    }
```

Lab task 5: Create a class to represent Animals. Animals have two behaviors; they can speak() and they can move(). An animal has a name and location (x and y coordinate). By default, when an animal moves, the text "This animal moves forward" is displayed. By default, when an animal speaks, the text "This animal speaks" is displayed. A general Animal should not be able to be instantiated.

Eagle and Tiger are also Animals. When an Eagle moves and Tiger behave such that where "animal" is displayed in speak() or move(), "eagle" or "tiger" is displayed by the appropriate classes.

Create a Client class and write the required code to show polymorphic calls to speak() and move() methods.

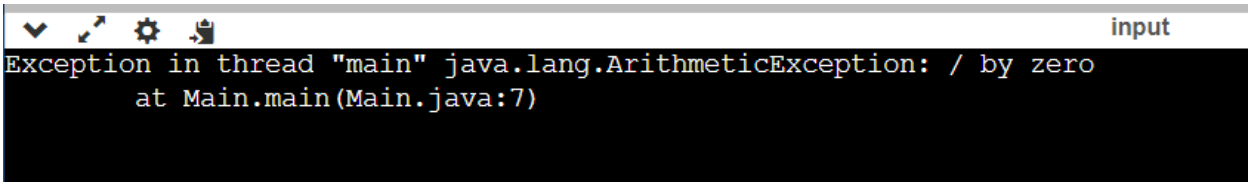
Lab 15

Part a: Exception Handling in JAVA

An Exception can be anything that interrupts the normal flow of the program. When an exception occurs program processing gets terminated and doesn't continue further. In such cases, we get a system-generated error message. The good thing about exceptions is that they can be handled. Exceptions are conditions within the code. A developer can handle such conditions and take necessary corrective actions.

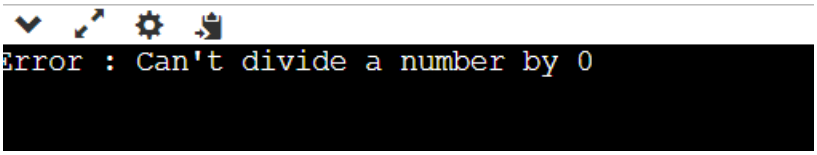
Example – Without Exception Handling

```
public class Main
{
	public static void main(String[] args) {
		int num1, num2;

		num1 = 0;
		num2 = 62/num1;
		System.out.println("The result is : " +num2);

	}
}
```



Example – With Exception Handling

```
public class Main
{
	public static void main(String[] args) {
		int num1, num2;
		try{
		   num1 = 0;
		   num2 = 62/num1;
		   System.out.println("The result is : " +num2);

	}

	catch(ArithmeticException e){
	   System.out.println("Error : Can't divide a number by 0");
	}

	}
}
```



Example 2 // Implementation of Multiple Catch Blocks

Main.java

```
1  public class Main
2  {
3      public static void main(String[] args) {
4          try{
5              int a[] = new int[7];
6              a[4] = 30/0;
7              System.out.println("First print statement in try block");
8          }
9
10         catch(ArithmeticException e){
11             System.out.println("Warning :  Arithmetic Exception");
12
13         }
14
15         catch(ArrayIndexOutOfBoundsException e){
16             System.out.println("Warning :  Array Index Out of Bound Exception");
17         }
18
19         catch(Exception e){
20             System.out.println("Warning :  Some other Exception");
21         }
22
23         System.out.println("Out of try-catch block");
24 }
25 }
26
```

input

```
Warning :  Arithmetic Exception
Out of try-catch block
```

Exercise:

Q1: Write a program that prompts the user to read two integers and displays their sum. Your program should prompt the user to read the number again if the input is incorrect.

Main.java

```java
import java.util.Scanner;
import java.util.InputMismatchException;
public class Main
{
    public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
        int n1 = 0;
        int n2 = 0;
        while(true) {
            System.out.print("Please enter in 2 integers to get their sum: ");
            try {
                n1 = in.nextInt();
                n2 = in.nextInt();

                break;
            }
            catch (InputMismatchException e) {
               System.out.println("You must enter 2 integers");
               in.nextLine();
            }

        }
        System.out.println("The sum is "+(n1+n2));
    }

}

```

input

```
You must enter 2 integers
Please enter in 2 integers to get their sum: 5
9
The sum is 14
```

Q2: Write a program that meets the following requirements:

■ Create an array with 100 randomly chosen integers.

■ Prompts the user to enter the index of the array, and then displays the corresponding element value. If the specified index is out of bounds, display the message Out of Bounds.

Main.java

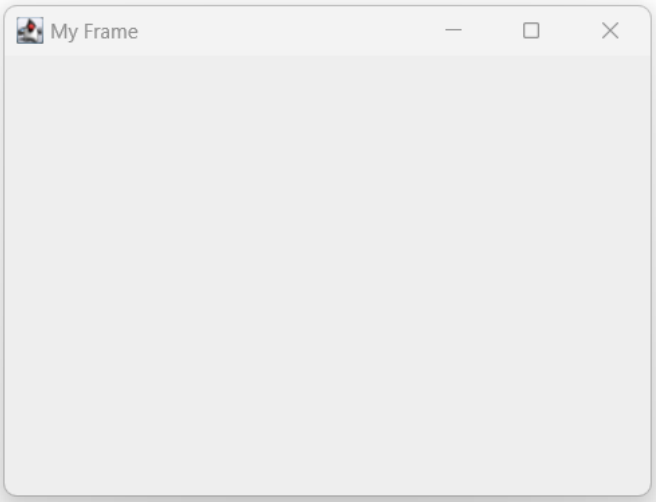
```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int[] array = getArray();
        System.out.print("Enter the index of the array: ");
        try {
            System.out.println("The corresponding element value is " +
                array[input.nextInt()]);
        }
        catch (ArrayIndexOutOfBoundsException ex) {
            System.out.println("Out of Bounds.");
        }
    }

    public static int[] getArray() {
        Random rand = new Random();
        int[] array = new int[100];
        for (int i = 0; i < array.length; i++) {
            array[i] = rand.nextInt(100);
        }
        return array;
    }
}

```

Part b: Graphical User Interface (GUI)

Example 1 // Creating a Frame

Frame.java

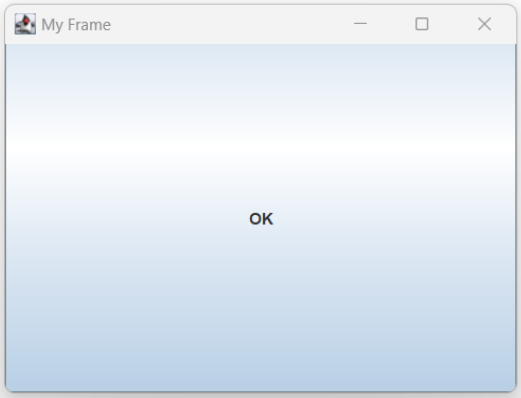
```java
import javax.swing.*;
public class Frame {
    public static void main(String[] args){
        JFrame frame = new JFrame("My Frame");
        frame.setSize(400,300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



Example 2 // Adding features (buttons) to the frame

Frame.java

```java
import javax.swing.*;
public class Frame {
    public static void main(String[] args){
        JFrame frame = new JFrame("My Frame");//create a frame

        //add a button to the frame
        JButton btn = new JButton("OK");
        frame.add(btn);

      //set size of frame
        frame.setSize(400,300);

        //center the frame
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //display the frame
        frame.setVisible(true);
    }
}
```



Frame.java | ShowFlowLayout.java

```java
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JFrame;
import java.awt.FlowLayout;

public class ShowFlowLayout extends JFrame {
public ShowFlowLayout() {
    setLayout(new FlowLayout(FlowLayout.LEFT, 10,20));
    add(new JLabel("First Name"));
    add(new JTextField(8));
    add(new JLabel("MI"));
    add(new JTextField(1));
    add(new JLabel("Last Name"));
    add(new JTextField(8));
}

public static void main(String[]args) {
    ShowFlowLayout frame = new ShowFlowLayout();
    frame.setTitle("Show Flow Layout");
    frame.setSize(400,200);
    frame.setLocationRelativeTo(null);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
}
}
```