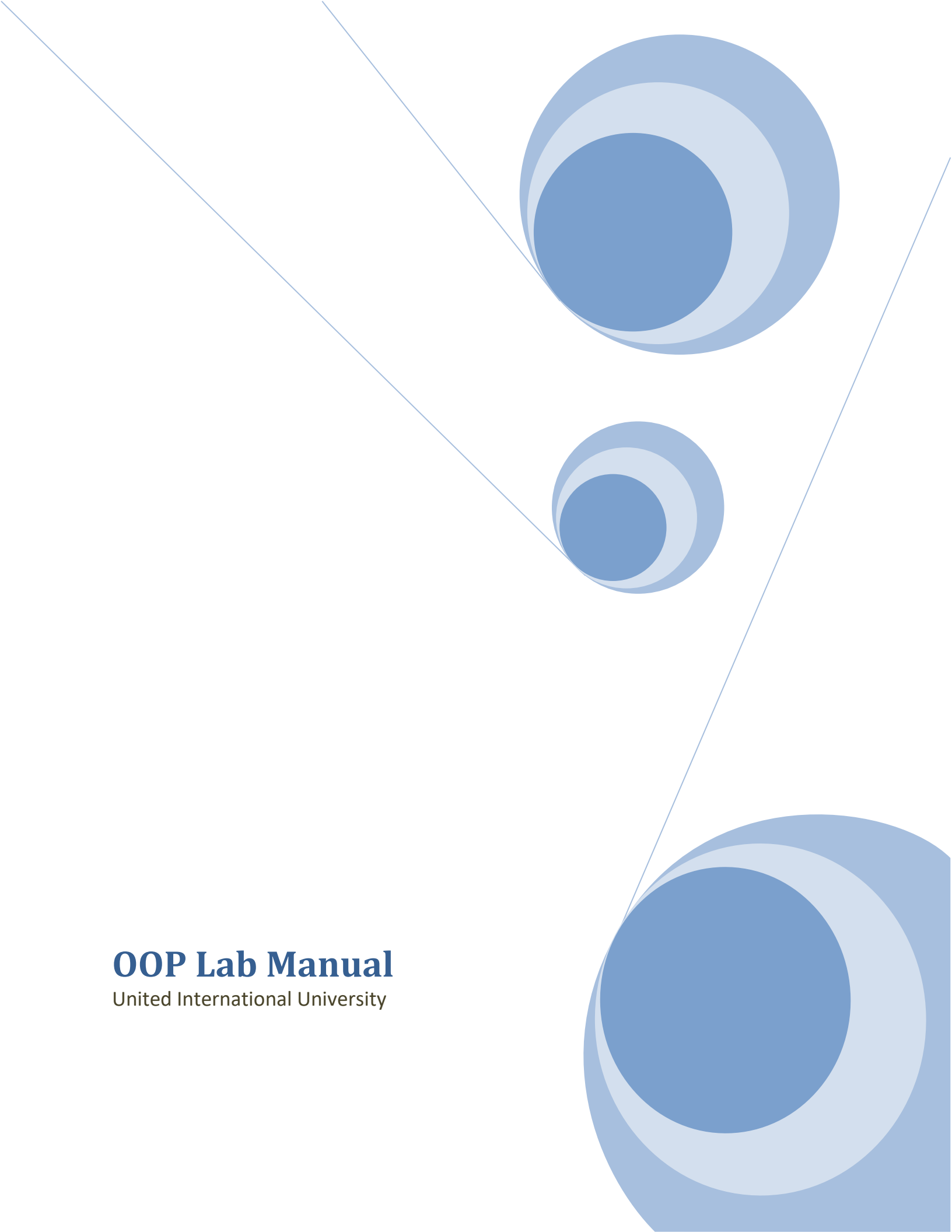


OOP Lab Manual

United International University

Table of Content

Introduction **4**
Course Outcome 4
Lab1 ..... **5**
Topics Covered 5
Introduction to Java 5
Tools Set-Up 5
Java Basics 6
Arrays 7
Sample Lab Exercises 8
Lab2 ..... **9**
Topics Covered 9
Introduction to OOP 9
Sample Lab Exercises 11
Lab3 ..... **12**
Topics Covered 12
Constructor 12
User Input 13
Sample Lab Exercises 14
Lab4 ..... **15**
Topics Covered 15
Inheritance 15
Method override 16
SubClass Polymorphism 16
Sample Lab Exercises 17
Lab5 ..... **18**
Topics Covered 18
Abstraction: 18

Sample Lab Exercises 20

Lab6 – Mid Exam 21

Mid Exam 21

Lab7 22

Topics Covered 22

Graphical User Interface (GUI) 22

Sample Lab Exercises 22

Lab8 23

Topics Covered 23

Exception 23

Input / Output (will use BufferedReader, BufferedWriter, PrintWriter, Scanner) 23

Sample Lab Exercises 23

Lab9 24

Topics Covered 24

Socket 24

Sample Lab Exercises 25

Lab10 26

Topics Covered 26

Thread 26

Collection 26

Sample Lab Exercises 27

Lab11 28

Topics Covered 28

GUI-Graphics 28

Sample Lab Exercises 28

Lab12-Presentation 30

Presentation on Project 30

Introduction

Course Outcome

- Describe the Object Oriented Programming Features.
- Able to analyze a problem and develop well designed applications using the OOP features.
- Use a modern/popular IDE to develop the application.
- Be able to use the Library effectively.
- Develop self-driven project using the concepts learned from course and their own research.

Lab1

Topics Covered

- Introduction to Java
- Tools Set-up
- Java Basics
    - Application Class
    - Naming convention
    - Programming Language Basics
    - Arrays

Introduction to Java

JAVA was developed by Sun Microsystems Inc in 1991, later acquired by Oracle Corporation. It was developed by James Gosling and Patrick Naughton.

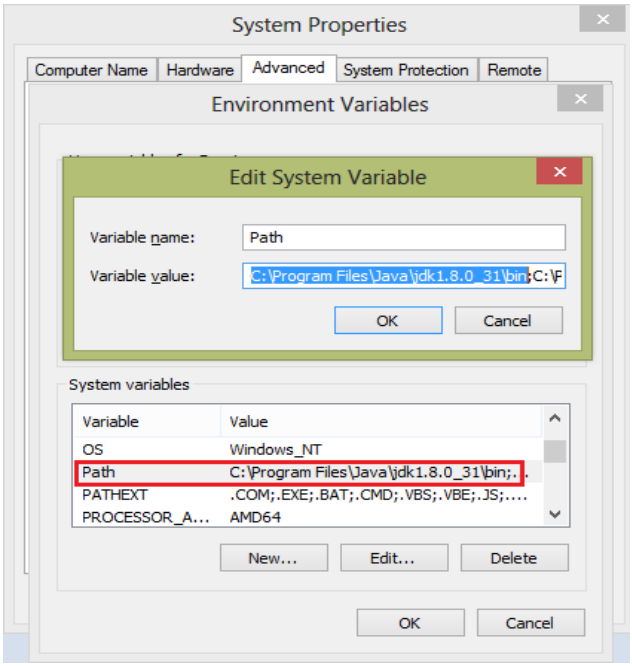
- Java is related to C++, which is a direct descendent of C.
    - Much of the character of Java is inherited from these two languages.

Tools Set-Up

Step1: Install Java and Path Set-up

- Need to install Java(**JDK and JRE**). Get the latest version from Java Standard Edition(SE) from http://www.oracle.com/technetwork/java/javase/downloads/index.html
    - After installing Java you need to set-up the "**Path**" environment variable which is available from **My Computer** under **Advanced Properties** tab.

Note: Do not delete anything in "Path" variable. Just add your path "C:\Program Files\Java\jdk1.8.0_31\bin;" (Depending on your version the path will change) at the beginning of the existing value.



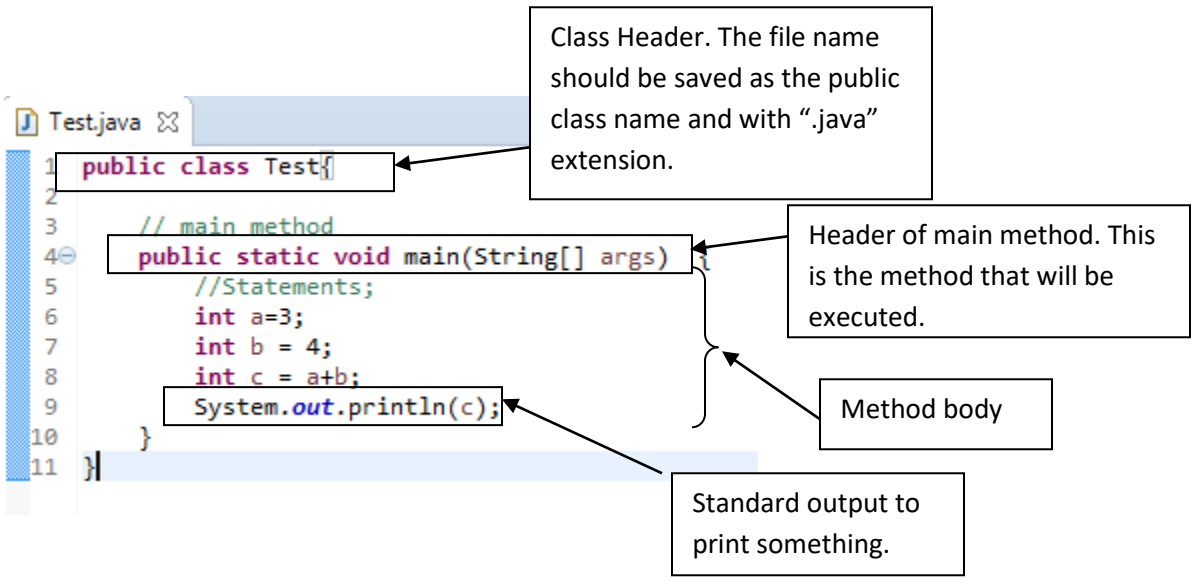
Step 2: Install IDE

- Need an IDE: Eclipse or NetBeans or IntelliJ IDEA.
- *You can install*
    - eclipse from : http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/mars1
    - NetBeans: http://www.oracle.com/technetwork/java/javase/downloads/index.html
    - IntelliJ IDEA: https://www.jetbrains.com/idea/download/#section=windows

Java Basics

Application Class

- The class that contains "main" method.
    - No stand alone functions in Java - **All code is part of a class**
- When an application is launched, the "main" method of this class gets executed.
- Example application class:



public static void main(String args[])

- **main -** the starting point for any java program
- **public** - to make this method accessible to all
- **static -** to call this method instantiation is not needed. JVM calls Classname.main
- **void -** the main method does not return anything
- **String args[] -** An array of string objects that holds the command line arguments passed to the application

System.out.print*ln*()

- Used to print a text
- **System -** is a class in Java API (Application Program Interface)

- **out -** represents the standard output, static member of System, part of API
- **println() –** method to print the text

Note:

- Java assumes that you will be using a GUI for input from the user Hence there is no "simple" way to read input from a user
- Later it is simplified by introducing “Scanner” class in Java Library – we will cover this topic later.

Naming Convention

- All java source file should end with .java
- Each .java file can contain **only one public class**
- The **name of the file** should be **the name of the public class** plus ".java"
- Do not use abbreviations in the name of the class
- If the class name contains **multiple words** then **capitalize the first letter of each word** ex. HelloWorld.java
- It’s a good practice to have one class per file

Programming Language Basics

- Data Types & Variables
- Operators
- Control Statement
    - If-else-if
    - Switch
    - Looping
        - for
        - while
        - do-while
    - break, continue

Arrays

- **Declaration and creating Array object**
  int[] sampleArray;
  sampleArray = new int[10];
  Or
  int[] sampleArray = new int[10];

- **Initialization**
    - During declaration
      int[] sampleArray = {1,2,3,4,5};
    - After declaration
      int[] sampleArray;
      sampleArray = new int[]{1,2,3,4,5};

Sample Lab Exercises

1. Write the “Hello World” program.

Steps to follow:

- Open the IDE.
- Open a new Project named “Hello World”
- Open a new Class named “HelloWorld” (no space in the name)
- Add the main method if it is not already added

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

}
```

- Add the following statement **inside the main method t**o print/display the text “Hello World!”

```
System.out.println(“Hello World!”);
```

- Save the file
- Now run the project from the menu bar or using the icon

2. Write a Java program to print the first 10 even numbers.

3. Write a Java program that will go through the items of an **array** and find the **max** and **min** value. Take the following values as the input of the array

{2, 3, 9, 8, 13, 1, 5, 19, 15, 0, 4}

Lab2

Topics Covered

- Introduction to OOP

Introduction to OOP

- All computer programs consist of two elements:
  - Logic (coded in functions) and data.
- In "C" we always think about action and implement the action using a function. Here the function takes input data, processes it, and produces output data.
- But in Object-oriented programming (OOP) model, program is organized around "objects" rather than "actions". Object is defined by a class.
- Class has 2 types of member
  - Fields/Attributes – represented by instance variables
  - Actions/Behaviors – represented by method

Structure/Format of Class

```
<modifier> class <class name>{
    // Declare attributes
    <modifier> <data type> <attribute name>;
    <modifier> <data type> <attribute name> [=<value >] ;

    // Constructor
    <modifier> <class name> (<argument>){
    <statements>;
    }

    // Method
    <modifier> <return type > <method name>(<argument>){
    <statements>;
    }
} // end of class
```

Example: - Student Class

```java
public class Student{
	// Instance variables
	public String name;
	public String id;
	public float cgpa;
	public int creditCompleted;

	// Constructor – we will discuss later
	public Student(String name, String id, float cgpa, int creditCompleted) {
		this.name = name;
		this.id = id;
		this.cgpa = cgpa;
		this.creditCompleted = creditCompleted;
	}
	// Methods
```

```
	public void updateCgpa(int credit, float gpa){
		cgpa = (cgpa*creditCompleted + credit*gpa)/(creditCompleted+credit);
		creditCompleted = creditCompleted + credit;
	}
	public float getCgpa(){
		return cgpa;
	}
}
```

Creating object and accessing members

```
public class TestMultipleStudents {
	public static void main(String[] args) {
		// Create object
		Student studentR = new Student();
		Student studentK = new Student();

		// Assign values to attributes
		studentR.name = "Rashid";
		studentR.cgpa = 3.0f;
		studentR.creditCompleted = 20;

		studentK.name = "Khaled";
		studentK.cgpa = 3.0f;
		studentK.creditCompleted = 20;

		// display cgpa before update
		System.out.println(studentR.name + "; Credit Completed: "+
studentR.creditCompleted +"; Previous cgpa: "+ studentR.cgpa);
		System.out.println(studentK.name + "; Credit Completed: "+
studentK.creditCompleted +"; Previous cgpa: "+ studentK.cgpa);

		// update cgpa
		studentR.updateCgpa(3, 4.0f);

		// display cgpa after update
		System.out.println("After Update");
		System.out.printf("%s; Credit Completed: %d; New cgpa: %.2f\n",
studentR.name, studentR.creditCompleted, studentR.cgpa);
		System.out.printf("%s; Credit Completed: %d; New cgpa: %.2f",
studentK.name, studentK.creditCompleted, studentK.cgpa);
		}
}
```

Each Object has its own copy of the fields defined in the class.

studentK and studentR has different values for their name, id, cgpa, creditCompleted.

Output:

```
Rashid; Credit Completed: 20; Previous cgpa: 3.0
Khaled; Credit Completed: 20; Previous cgpa: 3.0
After Update
Rashid; Credit Completed: 23; New cgpa: 3.13
Khaled; Credit Completed: 20; New cgpa: 3.00
```

Sample Lab Exercises

1. Create a class named "**Box"** which has 3 attribute: ***length***, ***width***, ***height*** and a method named ***getVolume(). getVolume()*** method will calculate the volume of the Box and return the value. From "**main"** method create **2 Box objects** with different length, width, height, then call the getVolume() method and print the volumes.
2. Create a "**BankAccount"** class which has 3 attributes: ***name***, ***accountId***, and ***balance***. The class also has 4 methods "***void deposit(double amount)"***, "***void withdraw(double amount)"," double getBalane()"*** *and* ***"void display()".*** Now from the main method, create 2 BankAccount objects and call/access each of the methods.
3. Create an Address Book application, where a user can create new record, update record, delete record.

Lab3

Topics Covered

- Class/Object, Constructor,
- package, access modifier
- getter/setter
- Array (Reference Type)
- User input
    - Scanner
    - JOptionPane (GUI)

Constructor

- A *constructor*
    - *Allocate space for instance variables.*
    - *Initializes an object (its instance variables) immediately upon creation.*
- *Syntax:*
    - *It has the **same name as** **the class***.
    - Syntactically similar to a method. Except has **no return type**. Not even **void**.
        - This is because the **implicit return type of a class' constructor is a reference of class type itself.**
- If no constructor is specified in a class, Java provides a parameter-less default constructor.

Example

```
class Student{
    // Instance variables
    public String name;
    public String id;
    public float cgpa;
    public int creditCompleted;

    public Student(String name, String id, float cgpa, int creditCompleted) {
        this.name = name;
        this.id = id;
        this.cgpa = cgpa;
        this.creditCompleted = creditCompleted;
    }
    // Methods
    public void updateCgpa(int credit, float gpa){
        cgpa = (cgpa*creditCompleted + credit*gpa)/(creditCompleted+credit);
        creditCompleted = creditCompleted + credit;
    }
    public float getCgpa(){
        return cgpa;
    }
}
```

Constructor

Without constructor/ With default constructor

```
public class TestStudent {

public static void main(String[] args) {
  // Create object
  Student student = new Student();

  // Assign values to attributes
  student.name = "Rashid";
  student.id = "011153001";
  student.cgpa = 3.0f;
  student.creditCompleted = 50;

  // display cgpa before update
  System.out.println("Credit Completed: "+
student.creditCompleted +"; Previous cgpa: "+
student.cgpa);

  // Calling method - update cgpa
  student.updateCgpa(3, 4.0f);
  student.updateCgpa(3, 4.0f);
  student.updateCgpa(3, 4.0f);

  // display cgpa after update
  System.out.printf("Credit Completed: %d;
New cgpa: %.2f",
student.creditCompleted,student.cgpa);
  }
}
```

With explicit constructor

```
public class TestStudent {

public static void main(String[] args) {
  // Create object

  Student student = new Student("Rashid",
"011153001", 3.0f, 50);

  // display cgpa before update
  System.out.println("Credit Completed: "+
student.creditCompleted +"; Previous cgpa: "+
student.cgpa);

  // Calling method - update cgpa
  student.updateCgpa(3, 4.0f);
  student.updateCgpa(3, 4.0f);
  student.updateCgpa(3, 4.0f);

 // display cgpa after update
  System.out.printf("Credit Completed: %d; New
cgpa: %.2f",
student.creditCompleted,student.cgpa);

}
```

User Input

Example:

- Code to read user input using Scanner:(need to import java.util.Scanner)

```
Scanner scan = new Scanner(System.in);
int intInput = scan.nextInt(); //read the next input and convert it to integer
double dInput = scan.nextDouble(); // read the next input as double
String sInut = scan.next(); //read the next input as String
String lInput = scan.nextLine(); // read the whole line as input
scan.close();
```

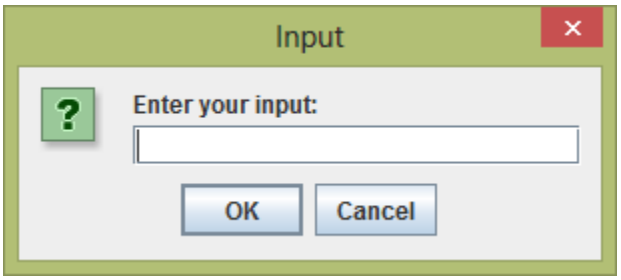
- Code to read user input using JOptionPane: (need to import javax.swing.JOptionPane)

```
String input = JOptionPane.showInputDialog(null, "Enter your input:"); // always
read the input as String

// Need to convert to appropriate type before using it
int intInput = Integer.parseInt(input); // Convert String to integer
```

```
double dInput = Double.parseDouble(input); // Convert String to double
```

Output:



Sample Lab Exercises

1. Write a Java application which will prompt user to provide a number as input. Read input as number and display the square of the number.
2. Write a Java application which will prompt user to provide two numbers as input. Read inputs, calculate the sum of those 2 numbers and display the result.
3. Update the "**BankAccount**" class you created at lab#2. Add couple constructors one without parameter ***"BankAccount()"*** with an empty body and the other with parameters ***"BankAccount(String name, String id, double balance)"***. From main method create 2 objects of BankAccount class using 2 different constructors. Call the display method and observe the outputs. What do you see?
4. Create a book store application which will help a book store owner to keep the record of its books, show all available books, sell books (should be able to sell multiple copies), and order new/existing books from publishers.

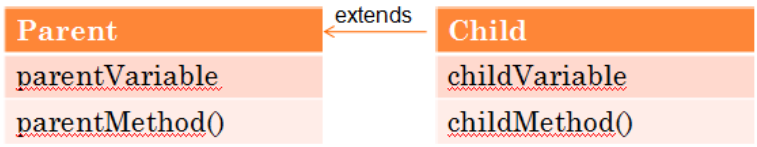
Lab4

Topics Covered

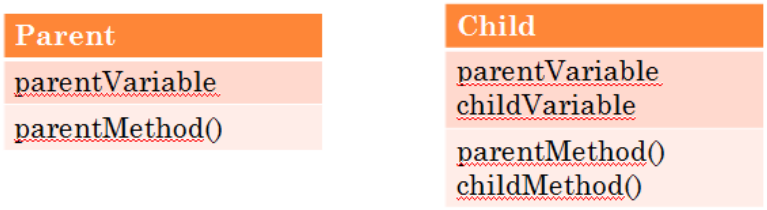
- Inheritance
- Method override
- SubClass Polymorphism

Inheritance

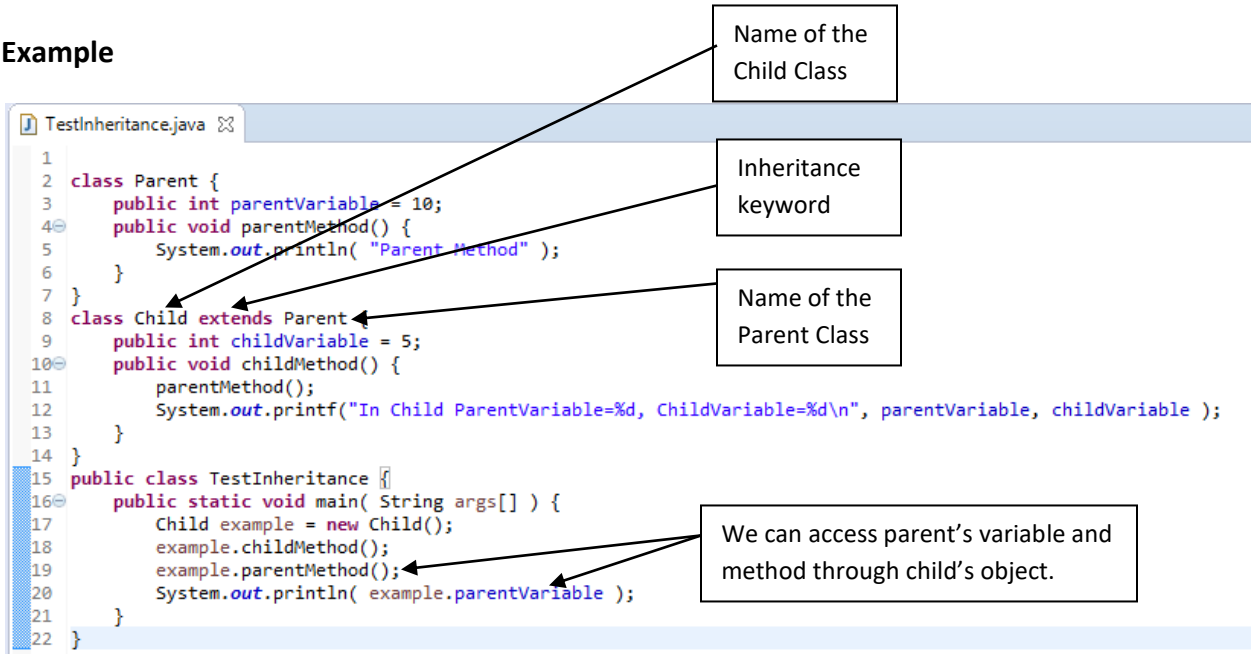
- Inheritance is the process by which one object acquires the properties of another object.
- It is a way to form new classes using classes that have already been defined.
- The new classes (child classes), take over (or inherit) attributes and behavior of the pre-existing classes (parent classes).



Act like



Example



Output:

```
Parent Method
In Child ParentVariable=10, ChildVariable=5
Parent Method
10
```

Method override

- A class replacing an ancestor's implementation of a method with an implementation of it own.
- When overriding a method in child class
    - Method Signature(name and argument list) and return type must be the same as the parent method.
    - Child method could be equal or more accessible.

Example:

TestInheritance.java

```
1
2  class Parent {
3      public int parentVariable = 10;
4      public void parentMethod() {
5          System.out.println( "Parent Method" );
6      }
7  }
8  class Child extends Parent {
9      public int childVariable = 5;
10     public void childMethod() {
11         parentMethod();
12         System.out.printf("In Child ParentVariable=%d, ChildVariable=%d\n", parentVariable, childVariable );
13     }
14
15     @Override
16     public void parentMethod() {
17         System.out.println( "Overriden Method" );
18     }
19
20 }
21 public class TestInheritance {
22     public static void main( String args[] ) {
23         Child example = new Child();
24         example.childMethod();
25         example.parentMethod();
26         System.out.println( example.parentVariable );
27     }
28 }
```

Parent class's method

Child class re-implemented Parent's method

Child's method get executed instead of parent's method.

Output:

```
Overriden Method
In Child ParentVariable=10, ChildVariable=5
Overriden Method
10
```

SubClass Polymorphism

- A **parent class reference** is used to **refer** to a **child class object**.
- Couple things to remember:

- The only possible way to access an object is through a reference variable.
- The type of the reference variable would determine the methods that it can invoke on the object.

- Using subclass polymorphism we can call or execute the child-class overriding method by the parent-class object

Example

TestInheritance.java

```
1  package pack1;
2
3  class Parent {
4      public void parentMethod() { System.out.println( "Parent Method" ); }
5  }
6  class Child extends Parent {
7      @Override
8      public void parentMethod() { System.out.println( "Child's Overriden Method" ); }
9  }
10
11 class OtherChild extends Parent {
12     @Override
13     public void parentMethod() { System.out.println( "Other child's Overriden Method" ); }
14 }
15
16 class GrandChild extends Child{
17     @Override
18     public void parentMethod() { System.out.println( "Grand child's Overriden Method" ); }
19 }
20 public class TestInheritance {
       public static void main( String args[] ) {
           Parent p  = new Parent();
           Parent c  = new Child();
           Parent oc = new OtherChild();
           Parent gc = new GrandChild();

           // Calling parentMethod with all these ref variable
           p.parentMethod();
           c.parentMethod();
           oc.parentMethod();
           gc.parentMethod();
       }
33 }
```

All 4 types of objects are referenced by Parent type variable.

Notice the output: During execution the method of the class the object belong to get executed.

Output

```
<terminated> TestInheritance (1) [Java Applica
Parent Method
Overriden Method
Other child's Overriden Method
Grand childs Overriden Method
```

Sample Lab Exercises

1. Create an Employee record system for a company. The application will help the company to view record of a specific employee, update his info. The Company has 4 types of employee (Salaried, Hourly, Commission, and Base plus commission), your application must handle all types of employee.
2. Create a **Banking System**, where a user can **create new account**, **deposit** money, **withdraw** money and **check** the balance. The application must handle different types of account e.g. "Savings Account", "Current Account" or a "Student Account"

Lab5

Topics Covered

- Abstraction
  - Abstract Class
  - Interface
- ArrayList
- static keyword

Abstraction:

- Abstraction is a process of hiding the implementation details from the user, **only the functionality will be provided** to the user.
- In other words, the user **will have the information on what the object does** instead of **how** it does it.
- In Java, abstraction is achieved using
  - Abstract classes and
  - interfaces

Abstract Class

An abstract class is a class that is incomplete, in that it describes a set of operations, but is missing the actual implementation of these operations. Abstract classes:

- Cannot be instantiated.
- So, can only be used through inheritance.

Example: Abstract Class

```
abstract class Animal{
	// instance variables
	String name, color;
	double weight;

	// Constructors
	Animal(){ }
	Animal(String name, String color, double weight){
		this.name = name;
		this.color = color;
		this.weight = weight;
	}

	Animal(String name, String color){ this(name,color, 0.0); }

	// Concrete methods
	public void eat(){ System.out.println(name + " eats."); }

	// abstract methods
	public abstract void makeSound();
}
```

```
class Bird extends Animal{
	public Bird() { name = "Bird";   }

	@Override
	public void makeSound() { System.out.println("Chirp");     }
}

class Tiger extends Animal{
	public Tiger() { name = "Tiger"; }

	@Override
	public void makeSound() { System.out.println("Roar"); }
}

public class TestAnimal {
	public static void main(String[] args) {
		Animal b = new Bird();
		Animal t = new Tiger();
		b.eat();
		t.eat();
		b.makeSound();
		t.makeSound();
	}
}
```

Output:

```
<terminated> TestAnimal [Jav
Bird eats.
Tiger eats.
Chirp
Roar
```

Interface

- Using interface, you can specify what a class must do, but not how it does it.
  - Interfaces can specify public methods but might not have the implementation of methods.
- Once it is defined, *any number of classes can implement an interface.*
- Also, one class can implement any number of interfaces.

Example: Interface

```
interface Flyable{
	String media = "Sky";

	void fly();
	boolean needFuel();
}

class Bird implements Flyable{
	@Override
	public void fly(){ System.out.printf("Bird can fly in the %s\n", Flyable.media);}

	@Override
	public boolean needFuel() { return false;     }
}

class Airplane implements Flyable{
	@Override
```

```
	public void fly(){ System.out.printf("Plane can fly in the %s\n", Flyable.media);}

	@Override
	public boolean needFuel() { return true; }
}

public class TestInterface {
	public static void main(String[] args) {
		Bird b = new Bird();
		Airplane a = new Airplane ();
		a.fly();
		b.fly();
	}
}
```

Sample Lab Exercises

1. Create an abstract class named Shape that contains two instance variables “dim1” and “dim2” of integer type and an abstract method named printArea(). Develop classes named **Rectangle**, **Triangle** and **Circle** which will be the subclasses of **Shape** class. Override the printArea() method and prints the area of the given shape.

Lab6 – Mid Exam

Mid Exam

Lab7

Topics Covered

- Graphical User Interface(GUI)

Graphical User Interface (GUI)

- 2 packages
    - Swing (javax.swing.* package)
    - AWT (java.awt.* package)

- Consists of multiple parts
    - Containers
    - Components
    - Events
    - Graphics

Sample Lab Exercises

1. Write a simple Java Swing/AWT based application for Counter. The UI contains a Label (“Counter”), a non-editable TextField and 2 Buttons (“Count” and “Reset”). TextField will show the value of the counter. Clicking the “Count” Button will increase the value of the counter by 1and displays the value in the TextField. Clicking the “Reset” button will reset the TextFeld value back to 0.

2. Make the TextField of problem#1 editable and add the following
    a. If user enters a number to the TextField, clicking the Button will increase the number by 1 and show the value.

3. Create a GUI application where clicking a button will check/uncheck a CheckBox and also change the text of the button. Clicking the button will
    b. Check/Select the checkbox if it is not checked/selected. Also set the text of the Button to “UnCheck”.
    c. Uncheck/Unselect the checkbox if it is checked/selected. Also set the text of the Button to “Check”.

Lab8

Topics Covered

- Exception
- Input / Output (will use BufferedReader, BufferedWriter, PrintWriter, Scanner)

Exception

- An *exception* is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- Following are some scenarios where an exception occurs.
    - A user has entered an invalid data.
    - A file that needs to be opened cannot be found.
    - A network connection has been lost in the middle of communications or the JVM has run out of memory.
- We need to handle the exception using try-catch-finally block to avoid abnormal shutdown of the program.

Input / Output (will use BufferedReader, BufferedWriter, PrintWriter, Scanner)

- The java.io package – contains Stream classes to perform input and output (I/O) in Java.
- All these streams represent
    - An input source and
    - An output destination.
    - Supports many data such as primitives, Object, localized characters, etc.

Sample Lab Exercises

1. Write a Java application which will prompt user to provide two numbers as input. Read inputs, calculate the sum of those 2 numbers and display the result. If any of the input is a negative number, throw an InvalidParameterException.

2. Assume a file "**input.txt**" contains comma-separated numbers. Write an application to read the numbers from the "**input.txt**" file, parse those numbers, calculate the summation, and then write the result to a different file name "**output.txt**". Handle all Exceptions with proper try/catch block.

Lab9

Topics Covered

- Socket

Socket

Socket is the Java programming model to establish a two-way communication link between two programs running on the network.

Example: Server-side Code

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

public class ServerExample {
	public static void main(String[] args) {
		ServerSocket s = null;
		Socket s1 = null;
		try	{
			s = new ServerSocket(5401);
			s1 = s.accept();
			System.out.println("Connection Extablished");

			BufferedReader br = new BufferedReader(new
InputStreamReader(s1.getInputStream()));

			String inp = br.readLine();
			while (inp != null && !inp.equals("exit"))	{
				System.out.println("Read:" + inp );
				inp = br.readLine();
			}
			System.out.println("Read:" + inp );
			br.close();
			if (!s1.isClosed())
		   s1.close();
		}
		catch(IOException e) {
		}
	}
}
```

Example: Client-side Code

```
import java.io.*;
import java.net.Socket;

public class ClientExample {
	public static void main(String[] args) {
		Socket s1 = null;
		try{
			s1 = new Socket("localhost",5401);
			System.out.println("Client: Connection Extablished");
```

```
                    BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(s1.getOutputStream()));
                    for(int i = 0; i<5; i++)
                    {
                           bw.write("Hello:"+i);
                           Thread.sleep(200);
                           bw.newLine();
                           bw.flush();
                    }
                    bw.close();
                    if (!s1.isClosed())
                  s1.close();
              }catch(IOException | InterruptedException e)
              {             }
       }
}
```

Sample Lab Exercises

1. Write a socket program where a client and server will communicate with each other. Client will write to socket and Server will read from the socket.

Lab10

Topics Covered

- Thread
- Socket
- Collections

Thread

Thread is an independent path of execution through program code.

Example:

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

		// Start the thread
		t1.start();t2.start();t3.start();
		try {  Thread.sleep(30);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		System.out.println("Main completed");
	}
}

class ThreadJobText implements Runnable{
	@Override
	public void run() {
		for(int i=0; i<5; i++)    {

	System.out.println(Thread.currentThread().getName()
+":"+i);
			try {
				Thread.sleep(10);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}
}
```

Sample Output:

```
<terminated> TestThr
Thread-1:0
Thread-2:0
Thread-0:0
Thread-2:1
Thread-1:1
Thread-0:1
Thread-0:2
Thread-2:2
Thread-1:2
Main completed
Thread-2:3
Thread-0:3
Thread-1:3
Thread-1:4
Thread-0:4
Thread-2:4
```

Collection

- Collection (sometimes called a container) is an object that holds other objects that are accessed, placed, and maintained under some set of rules.

- Examples
    - Sets
    - List
    - Map

Sample Lab Exercises

1) Create a multithreaded program by using Runnable interface and then create, initialize and start three Thread objects from your class. The threads will execute concurrently and display from 0 to 100 in the format [thread name: number].

2) Write a program to take numbers as input from user, add to an ArrayList and HashSet, and then print the items in the List and Set. While running the application try to enter same numbers multiple time and watch the changes in list and set.

Lab11

Topics Covered

- GUI-Graphics

GUI-Graphics

- All graphics are drawn relative to a window.
- The origin of each window is at the top-left corner and is 0,0.
- Coordinates are specified in pixels.
- A graphics context is encapsulated by the **Graphics class.**
- **Two ways to obtain a** graphics.
  - It is passed to a method, such as **paint( )/paintComponent() or update( ), as an argument.**
  - It is returned by the **getGraphics( ) method of Component.**
- **Graphics class defines a number of methods that draw various** types of objects, such as **lines**, **rectangles**, and **arcs**.
- Objects can be drawn **edge-only** or **filled**.
- Objects are drawn in the **currently selected color**, which is **black** by default.

Example:

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class DrawCircle extends JFrame {
	int x = 90, y = 90, d=40;

	public DrawCircle() {
		setSize(200, 200);
		setVisible(true);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
	}

	public void paint(Graphics g)	{
		g.setColor(Color.RED);
		g.fillOval(x, y, d, d);
	}

	public static void main(String[] args) {
		new DrawCircle();
	}
}
```

Output

Sample Lab Exercises

1. Create a GUI application where you will draw a circle of diameter 30 at a point where user clicks the mouse.

2. Create an application where a circle will be drawn after every 20 milliseconds. Use timer to fire the drawing event at every 20 millisecond and draw at (x+20, y) location where x and y is the coordinate of upper left corner of the last drawn circle.

   Note: Timer uses ActionListener for event handling. See below for the code to start a timer.

```
Timer t = new Timer(20, this);
t.start();
```

3. Create a simple game

Lab12-Presentation

Presentation on Project