Topic Covered:
Input and Output Using JOptionPane Dialog Boxes, The Integer, Double, and Other Wrapper Classes, Inheritance in Java, super class, subclass, extends keyword and the super Keyword.

INPUT AND OUTPUT USING JOPTIONPANE DIALOG BOXES

Java provides the JOptionPane class for creating dialog boxes—those familiar pop-up windows that prompt the user to enter a value or notify the user of an error. The JOptionPane class is in the javax.swing package.

Useful Methods of the *JOptionPane* Class

Return value	Method name and argument list
String	**showInputDialog(Component parent, Object prompt)** *static* method that pops up an input dialog box, where *prompt* asks the user for input. Returns the characters typed by the user as a *String*.
void	**showMessageDialog(Component parent, Object message)** *static* method that pops up an output dialog box with *message* displayed.

THE INTEGER, DOUBLE, AND OTHER WRAPPER CLASSES

A wrapper class "wraps" the value of a primitive type, such as double or int, into an object. These wrapper classes define an instance variable of that primitive data type, and also provide useful constants and methods for converting between the objects and the primitive data types.

Useful Methods of the *Integer* Wrapper Class

Return value	Method name and argument list
int	**parseInt(String s)** *static* method that converts the *Strings* to an *int* and returns that value

Useful Methods of the *Double* Wrapper Class

Return value	Method name and argument list
double	**parseDouble(String s)** *static* method that converts the *Strings* to a *double* and returns that value

PROGRAM 1: Using the JOptionPane class for creating dialog boxes

```
import javax.swing.JOptionPane;
class Test {
public static void main(String a[]){
JOptionPane.showMessageDialog(null, "Hello");
String name;
name = JOptionPane.showInputDialog("What is your name?");
JOptionPane.showMessageDialog(null, "Hello "+ name);
System.out.println();
}
}
```

PROGRAM 2: Using parseInt() and the JOptionPane class for creating dialog boxes

```
import javax.swing.JOptionPane;
class Test {
public static void main(String arg[]){
int a,b;
a = Integer.parseInt(JOptionPane.showInputDialog("Enter 1st Value:"));
b = Integer.parseInt(JOptionPane.showInputDialog("Enter 2nd Value:"));
JOptionPane.showMessageDialog(null, a+" + " +b+ " = "+ (a+b));
System.out.println();
}
}
```

PROGRAM 3:

```
import javax.swing.JOptionPane;

class MainMenu {
int option;

public void mainMenu() {
do {
option = Integer.parseInt(JOptionPane.showInputDialog("SELECT FROM
THIS MENU" + "\n1.Calculator\n2.Convertor\n3.Marksheet\n4.Exit"
+ "\nEnter Your Choice "));

switch(option) {
case 1:
Calculator cl = new Calculator();
cl.menu();
break;

case 2:
JOptionPane.showMessageDialog(null,"Convertor");
break;

case 3:
JOptionPane.showMessageDialog(null,"Marks Sheet");
break;

case 4:
JOptionPane.showMessageDialog(null,"Thank You for Using");
break;

default:
JOptionPane.showMessageDialog(null,"Invalid Option");
}
}
while(option !=4);
}
}
```

```
class Calculator {
int n1,n2, option;
char ch;

public void menu() {

do {
option=Integer.parseInt(JOptionPane.showInputDialog("Calculator\n"
+ "\n1Add\n2.Sub\n3.Multi\n4.Division\n5.Back"));

if(option == 1) {
add();
}
else if(option == 2) {
sub();
}
else if(option == 3) {
mul();
}
else if(option == 4) {
div();
}
else if (option == 5) {
JOptionPane.showMessageDialog(null, "Thank you for using Calculator");
}
}while(option != 5);
}

public void init() {
n1=Integer.parseInt(JOptionPane.showInputDialog("Enter 1st Value "));
n2=Integer.parseInt(JOptionPane.showInputDialog("Enter 2nd Value "));
}

public void add() {
do {
init();
JOptionPane.showMessageDialog(null, n1+" + " + n2 + " = "+(n1+n2));
ch = JOptionPane.showInputDialog("Calculate Again [Y/N]").charAt(0);
}while(ch=='y' || ch == 'Y');
}

public void sub() {
do {
init();
JOptionPane.showMessageDialog(null, n1+" - " + n2 + " = "+(n1-n2));
ch = JOptionPane.showInputDialog("Calculate Again [Y/N]").charAt(0);
}while(ch=='y' || ch == 'Y');
}
```

```
public void mul() {
do {
init();
JOptionPane.showMessageDialog(null, n1+" x " + n2 + " = "+(n1*n2));
ch = JOptionPane.showInputDialog("Calculate Again [Y/N]").charAt(0);
}while(ch=='y' || ch == 'Y');
}

public void div() {
do {
init();
JOptionPane.showMessageDialog(null, n1+" / " + n2 + " = "+(n1/n2));
ch = JOptionPane.showInputDialog("Calculate Again [Y/N]").charAt(0);
}while(ch=='y' || ch == 'Y');
}

}

class Test {
public static void main(String[] args) {
MainMenu ob = new MainMenu();
ob.mainMenu();
}
}
```

EXERCISE 10-1:

Run program number 3 and see the results. Now create two more classes Converter and MarkSheet. (See Calculator class for the design of both classes)
In Converter, use any 4 options for conversion of your choice. For Example Kilogram to Pounds, Hours to Minuets or Degree Celsius to Degree Fahrenheit, etc. The program should show a menu for choice. When user selects the choice the input box should take value of unit from user and convert it accordingly. The program should ask if you want to convert another value.
In MarkSheet, take five subject of your choice and show in menu then when a user select a choice the program should take input of marks from user for each subject. Finally shows the total, percentage and grade of that student.

INHERITANCE IN JAVA

- Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.
- This also provides an opportunity to reuse the code functionality and fast implementation time. When creating a class, instead of writing completely new data members and member methods, the programmer can designate that the new class should inherit the members of an existing class.
- The idea of inheritance implements the "is a" relationship. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.
- Inheritance lets us organize related classes into ordered levels of functionality, called hierarchies. The advantage is that we write the common code only once and reuse it in multiple classes.
- A subclass inherits methods and fields of its superclass. A subclass can have only one direct superclass, but many subclasses can inherit from a common superclass.
- Inheritance implements the "is a" relationship between classes. Any object of a subclass is also an object of the superclass.
- All classes inherit from the Object class.
- To specify that a subclass inherits from a superclass, the subclass uses the extends keyword in the class definition, as in the following syntax:

  ```
  AccessModifier class CassName extends SuperClassName
  ```
- A subclass does not inherit constructors or private members of the superclass. However, the superclass constructors are still available to be called from the subclass, and the private fields of the superclass are implemented as fields of the subclass.
- To access private fields of the superclass, the subclass needs to use the methods provided by the superclass.
- To call the constructor of the superclass, the subclass constructor uses the following syntax:

  ```
  super ( argument list );
  ```
- If used, this statement must be the first statement in the subclass constructor.
- A subclass can override an inherited method by providing a new version of the method. The new method's API must be identical to the inherited method. To call the inherited version of the method, the subclass uses the super object reference using the following syntax:

  ```
  super.methodName ( argument list );
  ```
- Any field declared using the protected access modifier is inherited by the subclass. As such, the subclass can directly access the field without calling its method.
- In Multilevel Inheritance, a subclass will be inheriting a superclass and as well as the subclass also act as the superclass to other class. In below image, the class A serves as a superclass for the subclass B, which in turn serves as a superclass for the subclass C.

PROGRAM 4: A simple example of inheritance.

```
// Create a superclass.
class A {

int i, j;

A() {
i = 100;
j = 200;
}

void showij() {
System.out.println("i and j: " + i + " " + j);
}
}

// Create a subclass by extending class A.
class B extends A {
int k;

B() {
k = 300;
}

void showk() {
System.out.println("k: " + k);
}

void sum() {
System.out.println("i+j+k: " + (i+j+k));
}
}

class Test{
public static void main(String args[]) {
B ob = new B();
ob.showij();
ob.showk();
ob.sum();

}
}
```

PROGRAM 5: This program uses inheritance to extend Box

```
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 width, double height, double depth) {
this.width = width;
this.height = height;
this.depth = depth;
}

// 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 extended 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);
}
}
```

PROGRAM 6: Using super to overcome name hiding.

```
class A {
int i;
}

// Create a subclass by extending class A.
class B extends A {
int i; // this i hides the i in A

B(int a, int b) {
super.i = a; // i in A
i = b; // i in B
}
void show() {
System.out.println("i in superclass: " + super.i);
System.out.println("i in subclass: " + i);
}
}

class UseSuper {
public static void main(String args[]) {
B subOb = new B(1, 2);
subOb.show();

}
}
```

PROGRAM 7: Using super to call superclass constructor.

```
public class A {
     private int i;

     A(int a){
     i = a;
     }

     void showA() {
     System.out.println("i in superclass: " + i);
     }
}

public class B extends A  {

     int i;

     B(int a, int b) {
     super(a);
     i = b;
     }

     void showB() {
     System.out.println("i in subclass: " + i);
     }
}

public class Test {

     public static void main(String[] args) {

          B subOb = new B(100, 200);
          subOb.showA();
          subOb.showB();

     }
}
```