

# Revision

CSC240, SWE211: Object Oriented Programming – Fall 2023

Dr. Walaa El-Ashmawi and Dr. Osama El-Ghonimy

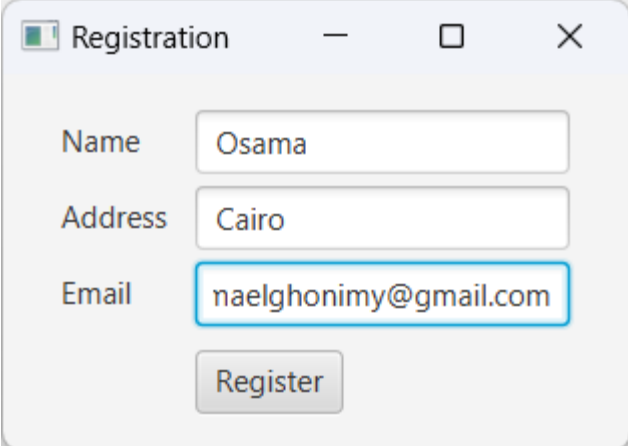


# ▼ ANNOUNCEMENTS

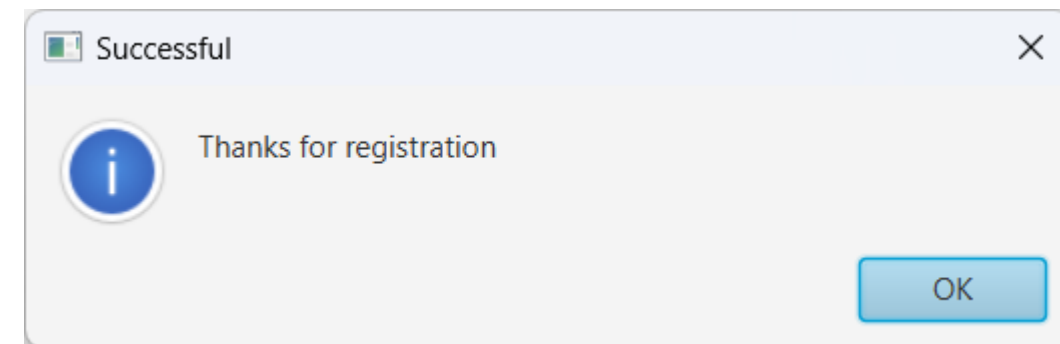
- Final exam:
  - Period: 90 minutes.
  - About 33% programming + about 67% other questions.
  - Computerized, NO COMPILER will be available.
  - Contents: All lectures including the revisions.
- Mini-project phase (2) discussion will be held after the final exam.  
Check the moodle for the schedule and grading criteria.

## ▼ EXAMPLE (1)

- Write a client-server application that collect student data during registration and send a student object to the localhost to be stored in a binary file there.
- The client is a GUI application which sends a student object when the user clicks a button and display a confirmation message.



A screenshot of a Windows-style application window titled "Registration". It contains three text input fields: "Name" with the value "Osama", "Address" with the value "Cairo", and "Email" with the value "naelghonimy@gmail.com". The "Email" field is highlighted with a blue border. Below the fields is a "Register" button.



## ▼ EXAMPLE (1)

- The server is a console application which continuously waiting for a client connection to append its sent student object to a binary file.
- The server should be able to serve multiple clients registering students at the same time from different locations.
- To prevent data loss, only one thread at a time should write to the binary file.

```
// Required imports
```

```
public class Server{  
    public static void main(String[] args) {  
        ServerSocket ss = new ServerSocket(8080);
```

```
}
```

```
}
```

```
// Required imports
```

```
public class Server{
    public static void main(String[] args) throws IOException {
        ServerSocket ss = new ServerSocket(8080);
        Socket s = ss.accept();

        ss.close();
    }
}
```

```
// Required imports
```

```
public class Server{
    public static void main(String[] args) throws IOException {
        ServerSocket ss = new ServerSocket(8080);
        Socket s = ss.accept();

                                                                    s.getInputStream()

        ss.close();        s.close();
    }
}
```

```
// Required imports
```

```
public class Server{  
    public static void main(String[] args) throws IOException {  
        ServerSocket ss = new ServerSocket(8080);  
        Socket s = ss.accept();  
        ObjectInputStream stream = new ObjectInputStream(s.getInputStream());  
  
        ss.close();        s.close();  
    }  
}
```



```
// Required imports
```

```
public class Server{  
    public static void main(String[] args) throws IOException {  
        ServerSocket ss = new ServerSocket(8080);  
        Socket s = ss.accept();  
        ObjectInputStream stream = new ObjectInputStream(s.getInputStream());  
  
        stream.readObject();  
  
        ss.close();        s.close();        stream.close();  
    }  
}
```

```
// Required imports
class Student {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        ServerSocket ss = new ServerSocket(8080);
        Socket s = ss.accept();
        ObjectInputStream stream = new ObjectInputStream(s.getInputStream());

        stream.readObject();

        ss.close();        s.close();        stream.close();
    }
}
```

```
// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        ServerSocket ss = new ServerSocket(8080);
        Socket s = ss.accept();
        ObjectInputStream stream = new ObjectInputStream(s.getInputStream());

        stream.readObject();

        ss.close();        s.close();        stream.close();
    }
}
```

```
// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        ServerSocket ss = new ServerSocket(8080);
        Socket s = ss.accept();
        ObjectInputStream stream = new ObjectInputStream(s.getInputStream());
        ObjectOutputStream file =
            new ObjectOutputStream(new FileOutputStream("objects.dat", true));
        Student st = (Student) stream.readObject();
        file.writeObject(st);

        ss.close();        s.close();        stream.close();
    }
}
```

```
// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        ServerSocket ss = new ServerSocket(8080);
        Socket s = ss.accept();
        ObjectInputStream stream = new ObjectInputStream(s.getInputStream());
        ObjectOutputStream file =
            new ObjectOutputStream(new FileOutputStream("objects.dat", true));
        Student st = (Student) stream.readObject();
        file.writeObject(st);

        ss.close();          s.close();          stream.close();          file.close();
    }
}
```

```
// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        ServerSocket ss = new ServerSocket(8080);
        Socket s = ss.accept();
        ObjectInputStream stream = new ObjectInputStream(s.getInputStream());
        ObjectOutputStream file =
            new ObjectOutputStream(new FileOutputStream("objects.dat", true));
        file.writeObject((Student) stream.readObject());

        // How to configure the server to continuously wait for client connections?
        ss.close();          s.close();          stream.close();          file.close();
    }
}
```

```
// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        ServerSocket ss = new ServerSocket(8080);
        while (true) {
            Socket s = ss.accept();
            ObjectInputStream stream = new ObjectInputStream(s.getInputStream());
            ObjectOutputStream file =
                new ObjectOutputStream(new FileOutputStream("objects.dat", true));
            file.writeObject((Student) stream.readObject());

            ss.close();          s.close();          stream.close();          file.close();
        }
    }
}
```

```
// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        ServerSocket ss = new ServerSocket(8080);
        while (true) {
            Socket s = ss.accept();
            ObjectInputStream stream = new ObjectInputStream(s.getInputStream());
            ObjectOutputStream file =
                new ObjectOutputStream(new FileOutputStream("objects.dat", true));
            file.writeObject((Student) stream.readObject());

            // How to serve multiple clients?
            s.close();      stream.close();      file.close();
        }
    }
}
```



```
// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        ServerSocket ss = new ServerSocket(8080);
        while (true) {
            Socket s = ss.accept();
            new Thread(()->{
                try {
                    ObjectInputStream stream = new ObjectInputStream(s.getInputStream());
                    ObjectOutputStream file =
                        new ObjectOutputStream(new FileOutputStream("objects.dat", true));
                    file.writeObject((Student) stream.readObject());
                    s.close();      stream.close();      file.close();
                } catch (IOException | ClassNotFoundException e){
                    System.out.println(e);
                }
            }).start();
        }
    }
}
```

```
// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException {
        ServerSocket ss = new ServerSocket(8080);
        while (true) {
            Socket s = ss.accept();
            new Thread(()->{
                try {
                    ObjectInputStream stream = new ObjectInputStream(s.getInputStream());
                    ObjectOutputStream file =
                        new ObjectOutputStream(new FileOutputStream("objects.dat", true));
                    file.writeObject((Student) stream.readObject());
                    s.close();      stream.close();      file.close();
                } catch (IOException | ClassNotFoundException e){
                    System.out.println(e);
                }
            }).start();
        }
    }
}
```

```
// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException {
        ServerSocket ss = new ServerSocket(8080);
        ObjectOutputStream file =
            new ObjectOutputStream(new FileOutputStream("objects.dat", true));
        while (true) {
            Socket s = ss.accept();
            new Thread(()->{
                try {
                    ObjectInputStream stream = new ObjectInputStream(s.getInputStream());
                    file.writeObject((Student) stream.readObject());
                    s.close();      stream.close();      file.close();
                } catch (IOException | ClassNotFoundException e){
                    System.out.println(e);
                }
            }).start();
        }
    }
}
```

```

// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException {
        ServerSocket ss = new ServerSocket(8080);
        ObjectOutputStream file =
            new ObjectOutputStream(new FileOutputStream("objects.dat", true));
        while (true) {
            Socket s = ss.accept();
            new Thread(()->{
                try {
                    ObjectInputStream stream = new ObjectInputStream(s.getInputStream());
                    file.writeObject((Student) stream.readObject());
                    s.close();      stream.close();
                } catch (IOException | ClassNotFoundException e){
                    System.out.println(e);
                }
            }).start();
        }
    }
}

```

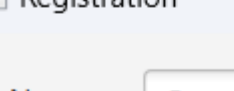
```
// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Server{
    public static void main(String[] args) throws IOException {
        ServerSocket ss = new ServerSocket(8080);
        ObjectOutputStream file =
            new ObjectOutputStream(new FileOutputStream("objects.dat", true));
        while (true) {
            Socket s = ss.accept();
            new Thread(()->{
                try {
                    ObjectInputStream stream = new ObjectInputStream(s.getInputStream());
                    synchronized(file) { file.writeObject((Student) stream.readObject()); }
                    s.close();      stream.close();
                } catch (IOException | ClassNotFoundException e){
                    System.out.println(e);
                }
            }).start();
        }
    }
}
```

```
// Required imports
```

```
public class Client extends Application{
    @Override
    public void start(Stage stage) {
        GridPane root = new GridPane();

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



Registration

Name Osama

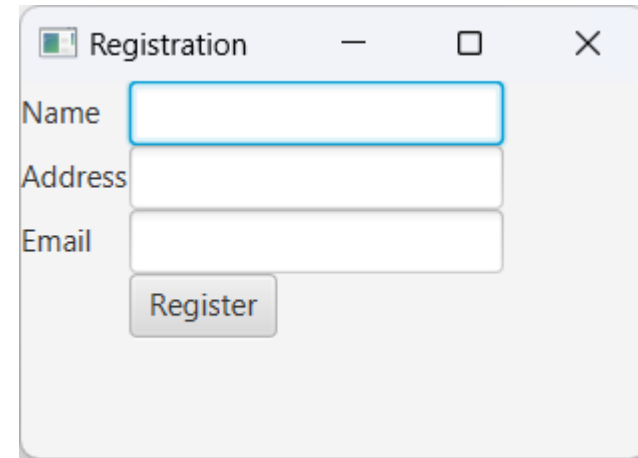
Address Cairo

Email naelghonimy@gmail.com

Register

```
// Required imports
```

```
public class Client extends Application{  
    @Override  
    public void start(Stage stage) {  
        GridPane root = new GridPane();  
        TextField tfAddress = new TextField();  
        root.add(new Label("Name"), 0, 0);  
        root.add(new Label("Address"), 0, 1);  
        root.add(new Label("Email"), 0, 2);  
        Button bt = new Button("Register");  
  
        stage.setScene(new Scene(root,250,150));  
        stage.setTitle("Registration");  
    }  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```



Registration

Name

Address

Email

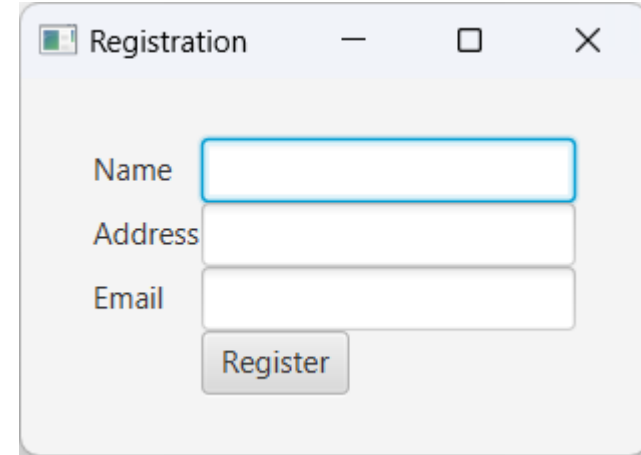
Register

```
TextField tfName = new TextField();  
TextField tfEmail = new TextField();  
root.add(tfName, 1, 0);  
root.add(tfAddress, 1, 1);  
root.add(tfEmail, 1, 2);  
root.add(bt, 1, 3);
```

```
stage.show();
```

```
// Required imports
```

```
public class Client extends Application{  
    @Override  
    public void start(Stage stage) {  
        GridPane root = new GridPane();  
        TextField tfAddress = new TextField();  
        root.add(new Label("Name"), 0, 0);  
        root.add(new Label("Address"), 0, 1);  
        root.add(new Label("Email"), 0, 2);  
        Button bt = new Button("Register");  
  
        stage.setScene(new Scene(root,250,150));  
        stage.setTitle("Registration");  
    }  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

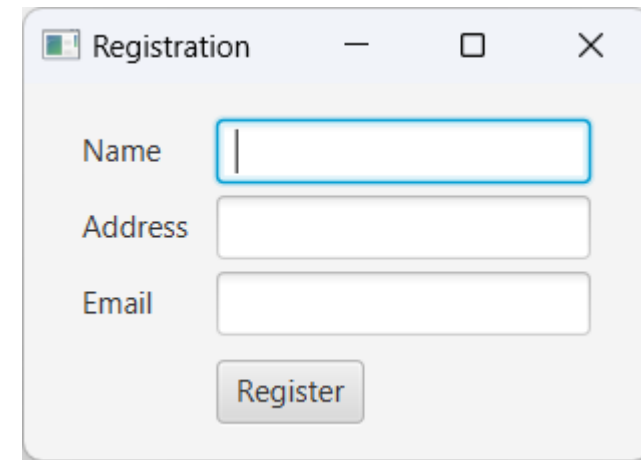


```
TextField tfName = new TextField();  
TextField tfEmail = new TextField();  
root.add(tfName, 1, 0);  
root.add(tfAddress, 1, 1);  
root.add(tfEmail, 1, 2);  
root.add(bt, 1, 3);  
  
root.setAlignment(Pos.CENTER);  
stage.show();
```



```
// Required imports
```

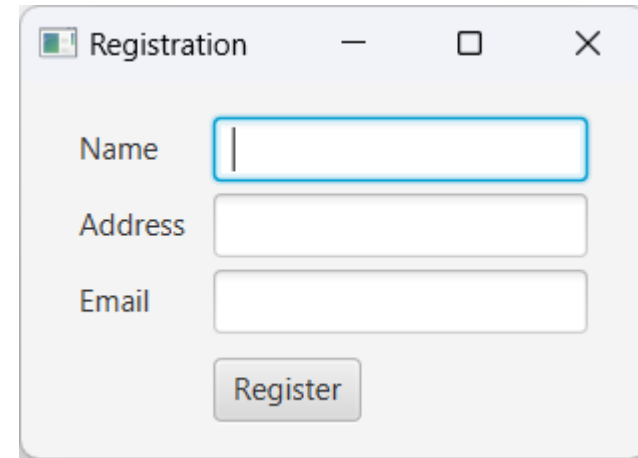
```
public class Client extends Application{  
    @Override  
    public void start(Stage stage) {  
        GridPane root = new GridPane();  
        TextField tfAddress = new TextField();  
        root.add(new Label("Name"), 0, 0);  
        root.add(new Label("Address"), 0, 1);  
        root.add(new Label("Email"), 0, 2);  
        Button bt = new Button("Register");  
        bt.setOnAction(e->{  
        });  
        root.setHgap(10);  
        stage.setScene(new Scene(root,250,150));  
        stage.setTitle("Registration");  
    }  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```



```
TextField tfName = new TextField();  
TextField tfEmail = new TextField();  
root.add(tfName, 1, 0);  
root.add(tfAddress, 1, 1);  
root.add(tfEmail, 1, 2);  
root.add(bt, 1, 3);  
  
root.setVgap(5);  
root.setAlignment(Pos.CENTER);  
stage.show();
```

```
// Required imports
```

```
public class Client extends Application{  
    @Override  
    public void start(Stage stage) {        ...  
        bt.setOnAction(e->{  
            try {  
                Socket s = new Socket("localhost", 8080);  
                ObjectOutputStream stream = new ObjectOutputStream(s.getOutputStream());  
  
                s.close();  
  
            } catch (IOException ex) {  
                System.out.println(ex);  
            }  
        });  
    }  
    ...  
}
```

A JavaFX-style registration window titled "Registration". It features three text input fields labeled "Name", "Address", and "Email". Below these fields is a "Register" button. The window has standard OS window controls (minimize, maximize, close) in the top right corner.

Registration

Name

Address

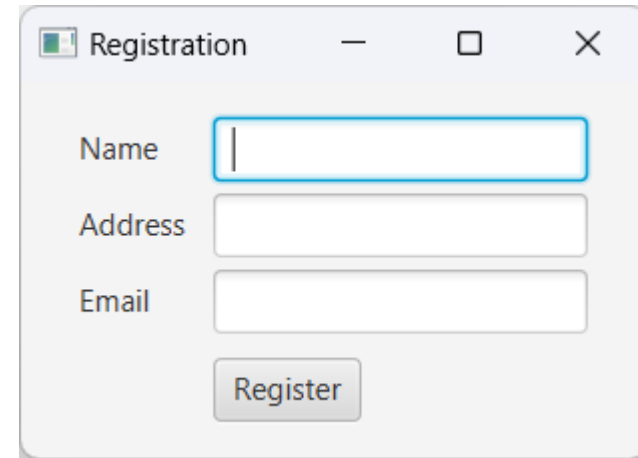
Email

```
// Required imports
```

```
public class Client extends Application{
    @Override
    public void start(Stage stage) {      ...
        bt.setOnAction(e->{
            try {
                Socket s = new Socket("localhost", 8080);
                ObjectOutputStream stream = new ObjectOutputStream(s.getOutputStream());

                stream.close();                s.close();

            } catch (IOException ex) {
                System.out.println(ex);
            }
        });
    }
    ...
}
```

A Java Swing window titled "Registration" with standard window controls (minimize, maximize, close). It contains three text input fields labeled "Name", "Address", and "Email". Below the fields is a "Register" button.

Registration

Name

Address

Email

Register

```

// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Client extends Application{
    @Override
    public void start(Stage stage) {      ...
        bt.setOnAction(e->{
            try {
                Socket s = new Socket("localhost", 8080);
                ObjectOutputStream stream = new ObjectOutputStream(s.getOutputStream());

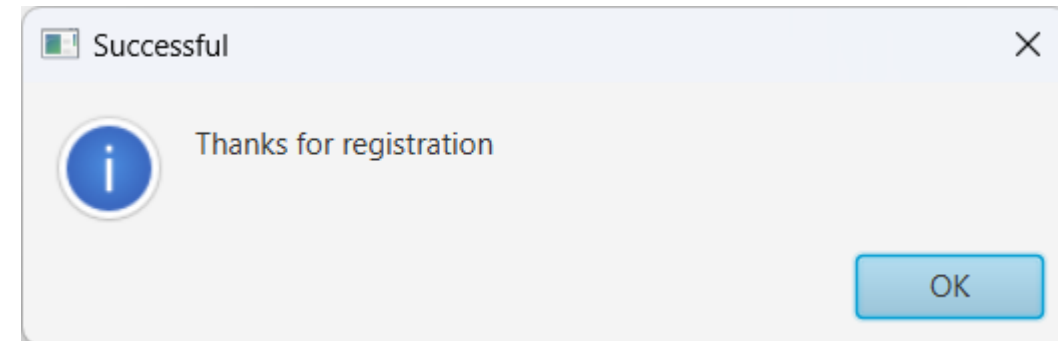
                new Student(tfName.getText(), tfAddress.getText(), tfEmail.getText())
                stream.close();
                                s.close();

            } catch (IOException ex) {
                System.out.println(ex);
            }
        });
    }
    ...
}

```

```
// Required imports
class Student implements Serializable {
    private String name, address, email;
    public Student(String n, String a, String e){ name = n; address = a; email = e; }
}

public class Client extends Application{
    @Override
    public void start(Stage stage) {      ...
        bt.setOnAction(e->{
            try {
                Socket s = new Socket("localhost", 8080);
                ObjectOutputStream stream = new ObjectOutputStream(s.getOutputStream());
                stream.writeObject(
                    new Student(tfName.getText(), tfAddress.getText(), tfEmail.getText()));
                stream.close();                                s.close();
                Alert ad = new Alert(Alert.AlertType.INFORMATION);
                ad.setTitle("Successful");                      ad.setHeaderText(null);
                ad.setContentText("Thanks for registration");   ad.show();
            } catch (IOException ex) {
                System.out.println(ex);
            }
        });
    }
    ...
}
```



# ▼ REMEMBER

- Text encryption example.

Input file contents (in.txt):  
Osama El-Ghonimy  
Spring 2023

Output file contents (out.txt):  
Ptbnb!Fm.Hipojnz  
Tqsjoh!3134

```
...
public class JavaTest{
    public static void main(String[] args) {
        try {
            Scanner in;
            in = new Scanner(new File("in.txt"));
            PrintWriter out;
            out = new PrintWriter("out.txt");
            while(in.hasNext()){
                String l = in.nextLine();
                String enc="";

                for (int i=0; i<l.length(); i++)
                    enc += (char)(l.charAt(i)+1);
                out.println(enc);
            }
            in.close();        out.close();
        } catch (FileNotFoundException e) {
            System.out.println(e);
        }
    }
}
```

## ▼ STRING CONCATENATION

- Strings are constant; their values cannot be changed after they are created. Concatenating strings using the + operator does not modify the strings involved, it creates a new string and assign it to the string reference.

```
...
public class JavaTest{
    public static void main(String[] args) {
        try {
            Scanner in;
            in = new Scanner(new File("in.txt"));
            PrintWriter out;
            out = new PrintWriter("out.txt");
            while(in.hasNext()){
                String l = in.nextLine();
                String enc="";

                for (int i=0; i<l.length(); i++)
                    enc += (char)(l.charAt(i)+1);
                out.println(enc);
            }
            in.close();        out.close();
        } catch (FileNotFoundException e) {
            System.out.println(e);
        }
    }
}
```

# ▼ STRINGBUILDER

- String represents fixed-length, immutable character sequences while StringBuilder represents growable and writable character sequences. It is faster than String when performing simple concatenations.

```
...
public class JavaTest{
    public static void main(String[] args) {
        try {
            Scanner in;
            in = new Scanner(new File("in.txt"));
            PrintWriter out;
            out = new PrintWriter("out.txt");
            while(in.hasNext()){
                String l = in.nextLine();
                StringBuilder enc =
                    new StringBuilder();
                for (int i=0; i<l.length(); i++)
                    enc += (char)(l.charAt(i)+1);
                out.println(enc);
            }
            in.close();        out.close();
        } catch (FileNotFoundException e) {
            System.out.println(e);
        }
    }
}
```



# ▼ STRINGBUILDER

- String represents fixed-length, immutable character sequences while StringBuilder represents growable and writable character sequences. It is faster than String when performing simple concatenations.

```
...
public class JavaTest{
    public static void main(String[] args) {
        try {
            Scanner in;
            in = new Scanner(new File("in.txt"));
            PrintWriter out;
            out = new PrintWriter("out.txt");
            while(in.hasNext()){
                String l = in.nextLine();
                StringBuilder enc =
                    new StringBuilder();
                for (int i=0; i<l.length(); i++)
                    enc.append(l.charAt(i)+1);
                out.println(enc);
            }
            in.close();        out.close();
        } catch (FileNotFoundException e) {
            System.out.println(e);
        }
    }
}
```

# ▼ STRINGBUILDER

## ○ Run. Output:

Input file contents (in.txt):

Osama El-Ghonimy

Spring 2023

Output file contents (out.txt):

Ptbnb!Fm.Hipojnz

Tqsjoh!3134

```
...
public class JavaTest{
    public static void main(String[] args) {
        try {
            Scanner in;
            in = new Scanner(new File("in.txt"));
            PrintWriter out;
            out = new PrintWriter("out.txt");
            while(in.hasNext()){
                String l = in.nextLine();
                StringBuilder enc =
                    new StringBuilder();
                for (int i=0; i<l.length(); i++)
                    enc.append(l.charAt(i)+1);
                out.println(enc);
            }
            in.close();          out.close();
        } catch (FileNotFoundException e) {
            System.out.println(e);
        }
    }
}
```

## ▼ EXAMPLE (3)

- Write a Java method that takes an array of different elements' type and return an array of only the ints.
- For example:
  - Input: [89, "hello", 12, 0.23, "java"]
  - Output: [89,12]

```
import java.util.ArrayList;
import java.util.Arrays;
public class JavaTest {
    public static int[] integers(Object[] in) {
        ArrayList<Integer> ints=new ArrayList<>();
        for(Object o : in)
            if(o instanceof Integer)
                ints.add((Integer)o);
        int[] out = new int[ints.size()];
        for(int i=0; i<ints.size(); i++)
            out[i] = ints.get(i);
        return out;
    }
    public static void main(String[] args) {
        Object[] a = {89,"hello",12,0.23,"java"};
        int[] ints = integers(a);
        System.out.println(Arrays.toString(ints));
    }
}
```

Output:  
[89, 12]

# INSTANCEOF

- instanceof  
evaluates to true if  
the left-hand-side  
object is a child of the  
right-hand-side class

```
class ParentClass {  
    // Class members  
}  
class ChildClass extends ParentClass {  
    // Class members  
}  
public class JavaTest {  
    public static void main(String[] args) {  
        ChildClass childObj = new ChildClass();  
        System.out.println(  
            childObj instanceof ParentClass);  
        System.out.println(  
            childObj instanceof Object);  
    }  
}
```

Output:

true  
true

# INSTANCEOF

- instanceof  
evaluates to true if  
the left-hand-side  
object is a child of the  
right-hand-side class  
or is the same type as  
the class provided.

```
class ParentClass {  
    // Class members  
}  
class ChildClass extends ParentClass {  
    // Class members  
}  
public class JavaTest {  
    public static void main(String[] args) {  
        ChildClass childObj = new ChildClass();  
        System.out.println(  
            childObj instanceof ParentClass);  
        System.out.println(  
            childObj instanceof Object);  
        System.out.println(  
            childObj instanceof ChildClass);  
    }  
}
```

Output:

true  
true  
true

# INSTANCEOF

- instanceof evaluates to true if the left-hand-side object is a child of the right-hand-side class or is the same type as the class provided.
- It also works with interfaces.

```
import java.io.Serializable;

class ChildClass implements Serializable{
    // Class members
}

public class JavaTest {
    public static void main(String[] args) {
        ChildClass childObj = new ChildClass();
        System.out.println(
            childObj instanceof Serializable);
    }
}
```

Output:  
true

## INSTANCEOF

- However, it gives a compile time error if it will be always false.

```
import java.io.Serializable;
import java.util.Scanner;

class ChildClass implements Serializable{
    // Class members
}

public class JavaTest {
    public static void main(String[] args) {
        ChildClass childObj = new ChildClass();
        System.out.println(
            childObj instanceof Serializable);
        System.out.println(
            childObj instanceof Scanner);
    }
}
```

Output:  
Compile time Error

## INSTANCEOF

- However, it gives a compile time error if it will be always false.
- A null reference is not an instanceof any class.

```
class ParentClass {
    // Class members
}
class ChildClass extends ParentClass {
    // Class members
}
public class JavaTest {
    public static void main(String[] args) {
        ChildClass childObj = null;
        System.out.println(
            childObj instanceof ParentClass);
        System.out.println(
            childObj instanceof Object);
        System.out.println(
            childObj instanceof ChildClass);
    }
}
```

Output:

false  
false  
false



## INSTANCEOF

- However, it gives a compile time error if it will be always false.
- A null reference is not an instanceof any class.
- Usually, it is used with different actual and declared types.

```
class ParentClass {  
    // Class members  
}  
class ChildClass extends ParentClass {  
    // Class members  
}  
public class JavaTest {  
    public static void main(String[] args) {  
        ParentClass obj = new ChildClass();  
        System.out.println(  
            obj instanceof ParentClass);  
        System.out.println(  
            obj instanceof Object);  
        System.out.println(  
            obj instanceof ChildClass);  
    }  
}
```

Output:

true  
true  
true

## INSTANCEOF

- However, it gives a compile time error if it will be always false.
- A null reference is not an instanceof any class.
- Usually, it is used with different actual and declared types.

```
class ParentClass {  
    // Class members  
}  
class ChildClass extends ParentClass {  
    // Class members  
}  
class OtherClass extends ParentClass {  
    // Class members  
}  
public class JavaTest {  
    public static void main(String[] args) {  
        ParentClass obj = new ChildClass();  
        System.out.println(  
            obj instanceof OtherClass);  
        // No compile time error  
    }  
}
```

Output:  
false

## ▼ EXAMPLE (4)

- Write a program to count the number of lines in a text file.
- How to count the number of words?

Output:

Number of lines = 2 lines

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class JavaTest {
    public static void main(String[] args) {

        File f = new File("file.txt");
        try {
            Scanner file = new Scanner(f);
            int c = 0;
            while(file.hasNext()){
                c++;
                file.nextLine();
            }
            System.out.print("Number of lines = ");
            System.out.println(c + " lines");
            file.close();
        } catch (FileNotFoundException e) {
            System.out.println(e);
        }
    }
}
```

## ▼ EXAMPLE (4)

- Write a program to count the number of lines in a text file.
- How to count the number of words?
- How to get file name as user input?

Output:

Number of words = 4 words

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class JavaTest {
    public static void main(String[] args) {
        System.out.print("Enter file name: ");
        Scanner s = new Scanner(System.in);
        File f = new File("file.txt");
        try {
            Scanner file = new Scanner(f);
            int c = 0;
            while(file.hasNext()){
                c++;
                file.next();
            }
            System.out.print("Number of words = ");
            System.out.println(c + " words");
            file.close();
        } catch (FileNotFoundException e) {
            System.out.println(e);
        }
    }
}
```

## ▼ EXAMPLE (4)

- Write a program to count the number of lines in a text file.
- How to count the number of words?
- How to get file name as user input?

Output:

Enter file name: file.txt  
Number of words = 4 words

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class JavaTest {
    public static void main(String[] args) {
        System.out.print("Enter file name: ");
        Scanner s = new Scanner(System.in);
        File f = new File(s.nextLine());
        try {
            Scanner file = new Scanner(f);
            int c = 0;
            while(file.hasNext()){
                c++;
                file.next();
            }
            System.out.print("Number of words = ");
            System.out.println(c + " words");
            file.close();
        } catch (FileNotFoundException e) {
            System.out.println(e);
        }
    }
}
```

# MCQ

Suppose the xMethod() is invoked from a main method in a class as follows,

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

xMethod() is \_\_\_\_\_ in the class.

- a. a static method**
- b. an instance method**
- c. a static method or an instance method**
- d. None of them**



# MCQ

If a variable is declared *static* in a particular class, what does that mean?

- a) There is only one copy of it that all instances of the class can access or share
- b) It cannot be changed
- c) It can only be changed by the instances of the class in which it is declared
- d) Each instance of the class has its own copy of the variable

Declare and construct an ArrayList with an initial capacity of 20 references to Object

- a) `ArrayList<Object> list = new ArrayList<Object>(20) ;`
- b) `ArrayList[Object] list = new ArrayList(20) ;`
- c) `ArrayList list[20] = new ArrayList() ;`
- d) `Object list(20) = new ArrayList() ;`



# MCQ

Which of these cannot be declared static?

- a) Object
- b) Constants
- c) Method
- d) variable

Which of the following statements are true?

- i. A subclass is a subset of a superclass.
  - ii. A subclass is usually extended to contain more functions and more detailed information than its superclass.
  - iii. "class A extends B" means A is a subclass of B.
  - iv. "class A extends B" means B is a subclass of A.
- a) ii and iii only
  - b) i only
  - c) iii only
  - d) i, ii and iii only
  - e) iv only





# MCQ

Given a class named **Student**, which of the following is a valid constructor declaration for the class?

- a) `Student (Student s) { }`
- b) `Student Student ( ) { }`
- c) `Static void Student(){ }`
- d) `Void Student ( ) { }`

Which option can we use in the try block to stop the execution of the “finally” block?

```
try {  
    // option  
} finally {  
    // code  
}
```

- a) `return`
- b) `break`
- c) `continue`
- d) No correct answer



# MCQ

What is printed as a result of executing the following code segment?

- a) [7, 2, 1, 2]
- b) [7, 1]
- c) [7, 2, 1]
- d) None of the mentioned

```
ArrayList p = new ArrayList();  
p.add(7);  
p.add(2);  
p.add(2);  
p.remove(2);  
p.add(1);  
p.add(2);  
System.out.println(p);
```

What is printed as a result of executing the following code segment?

- a) [1, 4, 5]
- b) [1, 2, 3, 4, 5]
- c) [2, 4, 5]
- d) [1, 4, 3, 5]

```
ArrayList aList = new ArrayList<Integer>();  
aList.add(1);  
aList.add(2);  
aList.remove(1);  
aList.add(1, 3);  
aList.set(1, 4);  
aList.add(5);  
System.out.println(aList);
```



Examine the following code:

```
ArrayList<String> list = new ArrayList<String>() ;  
  
list.add( "Andy" );  
list.add( "Bart" );  
list.add( "Carl" );  
list.add( "Doug" );  
list.add( "Elmo" );
```

# MCQ

Which of the following will change the list so that it looks like?

```
Andy  
Bart  
Carl  
Doug
```

- a) list.remove( list.size()-1 );
- b) list.remove( list.size() );
- c) list.remove( 5 );
- d) list.clear( "Elmo" );

What is the output of the following java code?

- a) [A,B,A,null]
- b) [A, B,A]
- c) [A,B,null]
- d) Compiler error

```
import java.util.ArrayList;  
class TestList {  
    public static void main(String[] args) {  
        ArrayList<String> al = new ArrayList<String>() ;  
        al.add("A");  
        al.add("B");  
        al.add("A");  
        al.add(null);  
        System.out.println(al);  
    }  
}
```



Analyze the following code.

# MCQ

```
public class Test {  
    public static void main(String args[]) {  
        NClass nc = new NClass();  
        nc.t = nc.t++;  
    }  
}  
  
class NClass {  
    int t;  
    private NClass() {  
    }  
}
```

- a) The program has a compile error because the NClass class has a private constructor.
- b) The program does not compile because the parameter list of the main method is wrong.
- c) The program compiles, but has a runtime error because t has no initial value.
- d) The program compiles and runs fine.



Analyze the following code.

# MCQ

```
public class Test {  
    public static void main(String[] args) {  
        B b = new B();  
        b.m(5);  
        System.out.println("i is " + b.i);  
    }  
}  
  
class A {  
    int i;  
  
    public void m(int i) { this.i = i; }  
}  
  
class B extends A {  
    public void m(String s) {  
    }  
}
```

- a) The method m is not overridden in B. B inherits the method m from A and defines an overloaded method m in B.
- b) The program has a compile error, because m is overridden with a different signature in B.
- c) The program has a compile error, because b.m(5) cannot be invoked since the method m(int) is hidden in B.
- d) The program has a runtime error on b.i, because i is not accessible from b.



# T/F

- Interfaces can be instantiated
- The modifiers public and static can be written in either order "public static" or "static public".
- Abstraction is supported by method overriding in java
- You can override a static method defined in a superclass
- You can write a constructor for an abstract class
- If **AbsClass** is an abstract class, the expression x instanceof AbsClass is always false.
- Classes, methods, and instance variables can all be declared **final**



## F Complete

- The general term for an object that represents the action of a user that may require a response is \_\_\_\_\_.

event or ActionEvent

- The general term for methods that are invoked as a result of a user action is \_\_\_\_\_.

event handler OR ActionEvent handler

- The method used to arrange for a button to notify another object when it is clicked later is to an \_\_\_\_\_.

addActionListener

- Which one of these ( list or set) allows duplicate elements?

List



F Write a program to repeat each character in a string once.

```
public class Ex2 {  
    public static void main(String a[]){  
  
        Scanner s=new Scanner(System.in);  
        System.out.println("Enter the text: ");  
        String x=s.nextLine();  
        System.out.println(doublechar(x));  
  
    }  
    private static String doublechar(String s) {  
        StringBuilder ns = new StringBuilder();  
  
        for( int i = 0; i<s.length();i++){  
            ns.append(s.charAt(i)).append(s.charAt(i));  
        }  
        return ns.toString();  
    }  
}
```

```
Enter the text:  
hello world  
hheellllloo  wwoorrlldd
```





## F Write java statements

- Set the 10 elements of integer array counts to zero

```
// declaring and setting 10 elements in the array with zero  
int array[]={0,0,0,0,0,0,0,0,0,0};
```

- Add one to each of the 15 elements of integer array bonus

```
int bonus[];  
bonus=new int[15];  
// declaring array bonus with 15 elements  
for(int i=0;i<15;i++){  
    bonus[i]+=1; }  
↔  
for(int i=0;i<bonus.length;i++){  
    bonus[i]+=1; }
```

```
int[] toys = { };  
toys.length → 0
```

- Declare a 2d array named “Arr\_2D” size n×n.

```
int[][] Arr_2D = new int[n][n];
```



Create a 2D array of size n with the following:

- Initialize it with a random integer numbers between 1 and 10.
- Set the diagonals to zeros

```
for( int i = 0 ; i < n ; i++ ) {  
    for ( int j = 0 ; j < n ; j++ ) {  
        if ( (i==j) || (i + j + 1) == n )  
            Arr_2D [i][j] = 0;  
        else  
            Arr_2D [i][j] = (int) (Math.random() * 10) + 1;  
//or Arr_2D [i][j] = random.nextInt(11);
```



## F What is the expected output?

```
public class Abc {  
    private int a = 10;  
    int b = 20;  
    protected int c = 30;  
    public int d = 40;  
  
    public void Abc() {  
        System.out.print(a);  
        System.out.print(b);  
        System.out.print(c);  
        System.out.print(d); }  
  
    public void xyz() {  
        System.out.println("Hello"); }  
  
    public static void main(String[] args) {  
        Abc obj = new Abc(); }  
}
```

No output because the default constructor has no something to do.



- F Create an object of HashMap type such that the key is of Integer type and the value is of String type and use an iterator to iterates over all elements of a HashMap.

keySet() → obtain all keys in the map

```
public class hashex
{
    public static void main ( String args [])
    {
        HashMap < Integer , String > hm = new HashMap < Integer , String >();
        hm . put (3 , " three ");
        hm . put (5 , " five ");
        if( hm . get (3) != null )
            System . out . println ("3 is present with value = " + hm . get (3));
        else
            System . out . println ("3 is absent ");

        if( hm . get (4) != null )
            System . out . println ("4 is present with value = " + hm . get (4));
        else
            System . out . println ("4 is absent ");

        Iterator < Integer > it = hm . keySet (). iterator ();
        while ( it . hasNext ())
        {
            System . out . println (" key = " + it . next () + " value = " + hm . get ( it . next () ));
        }
    }
}
```

3 is present with value = three  
4 is absent  
key = 3 value = five

3 is present with value = three  
4 is absent  
key = 3 value = three  
key = 5 value = five



- F Write a program to get the reference to the current thread by calling `currentThread()` method.

```
public class ThreadMethod {  
  
    public static void main(String args[])  
    {  
        Thread t = Thread.currentThread();  
        System.out.println ("current thread:" + t);  
        System.out.println("Name of the current thread:" + t.getName());  
        System.out.println ("Priority :" + t.getPriority());  
        t.setName("mythread");  
        System.out.println ("after name change :" + t);  
        t.setPriority (2);  
        System.out.println ("after priority change :" + t);  
        System.out.println ("number of active thread :" + t.activeCount());  
    }  
}
```

```
current thread:Thread[main,5,main]  
Name of the current thread:main  
Priority :5  
after name change :Thread[mythread,5,main]  
after priority change :Thread[mythread,2,main]  
number of active thread :1
```



- F Write a program to create two threads. In this class we have one constructor used to start the thread and run it. Check whether these two threads are run or not.

```
class CreateThread extends Thread
{
    String tname;
    Thread t;

    CreateThread(String s)
    {
        tname = s;
        t = new Thread(this, s);
        System.out.println ("New thread :" + t);
        t.start();
    }
    public void run()
    {
        try
        {
            for(int i = 5; i > 0; i--)
            {
                System.out.println (tname + ":" + i );
                Thread.sleep (500) ;
            }
        } catch (InterruptedException e) { }
        System.out.println(tname + " exiting...");
    }
}
```



```

public class ThreadMethod2
{
    public static void main(String args[])
    {
        CreateThread m1 =new CreateThread("one");
        CreateThread m2 = new CreateThread ("two");
        System.out.println("Thread m1 is alive :" + m1.t.isAlive());
        System.out.println ("Thread m2 is alive:" + m1.t.isAlive());
        try {
            System.out.println ("Waiting for thread to finish ....");
            m1.t.join();
            m2.t.join();
        } catch (InterruptedException e) { }
        System.out.println(" Thread m1 is alive :" + m1.t.isAlive());
        System.out.println(" Thread m2 is alive :" + m2.t.isAlive());
        System.out.println (" Main thread exiting ...");
    }
}

```

```

New thread :Thread[one,5,main]
New thread :Thread[two,5,main]
Thread m1 is alive :true
Thread m2 is alive:true
Waiting for thread to finish ....
one:5
two:5
one:4
two:4
two:3
one:3
two:2
one:2
two:1
one:1
two exiting...
one exiting...
Thread m1 is alive :false
Thread m2 is alive :false
Main thread exiting ...

```



- F Write a program to find the IP address of localhost and for any URL.

```
import java.net.InetAddress;
public class IP_Address
{
    public static void main(String args[]) throws Exception
    {
        InetAddress IP = InetAddress.getLocalHost();
        System.out.println("IP of my system is := "+IP.getHostAddress());
    }
}
```


IP of my system is := 127.0.0.1

```
public class IP_Address {
    public static void main(String args[])
        throws UnknownHostException
    {
        // The URL for which IP address needs to be fetched
        String s = "https://www.google.com.eg/";

        try {
            // Fetch IP address by getByName()
            InetAddress ip = InetAddress.getByName(new URL(s).getHost());

            // Print the IP address
            System.out.println("Public IP Address of: " + ip);
        }
        catch (MalformedURLException e) {
            // It means the URL is invalid
            System.out.println("Invalid URL");
        }
    }
}
```

Public IP Address of: www.google.com.eg/172.217.18.35





- F Write a server application which listens on given TCP port (port number 7). It reads string from client and sending back the same string to client like standard echo service.

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

public class echoServer {
    public final static int echoPort = 7;
    public static void main(String[] args) {
        ServerSocket theServerSocket;
        Socket theConnectionSocket;
        BufferedReader in;
        PrintWriter out;
        try {
            theServerSocket = new ServerSocket(echoPort);
            System.out.println("EchoServer ready at port "+ echoPort);
            while (true) {
                theConnectionSocket = theServerSocket.accept();
                try {
                    System.out.println("Request arrived!");
                    in = new BufferedReader(new
                        InputStreamReader(theConnectionSocket.getInputStream()));
                    out = new PrintWriter(theConnectionSocket.getOutputStream(),true);
                    while(true) {
                        String readText = in.readLine();
                        out.println(readText);
                    }
                }
                catch (IOException e) {
                    theConnectionSocket.close();
                }
            }
        }
        catch (IOException e) {
            System.err.println(e);
        }
    }
}
```



Test your  
knowledge

QUIZ  
time

```
public class ConstructorExample
{
    void ConstructorExample()
    {
        System.out.println("Learning");
    }
    ConstructorExample()
    {
        System.out.println("Java");
    }

    public static void main(String[] args)
    {
        ConstructorExample tc = new ConstructorExample();
    }
}
```

Java



# Test your knowledge

QUIZ  
time

```
class ConstructorExample
{
    int t = 1;
    ConstructorExample()
    {
        t = 2;
    }
}
public class Main
{
    public static void main(String args[])
    {
        ConstructorExample t1 = new ConstructorExample();
        System.out.println(t1.t);
    }
}
```

2



Test your  
knowledge

QUIZ  
time

```
public class ConstrcutorExample
{
    int a, b;
    public ConstrcutorExample(int x, int y)
    {
        a = x;
        b = y;
    }
    public static void main(String args[])
    {
        ConstrcutorExample p = new ConstrcutorExample();
    }
}
```

Compilation Error



Do you have  
any  
Questions?

