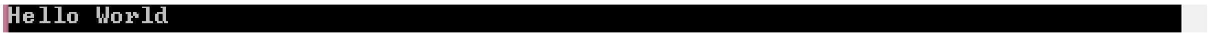
OCA 1: Declarations and Access Control

Exercises

Q1 – JAT1Ex1

Write a Java program, which displays the following message in the console, “Hello World”. Make use of the static imports feature to print the message.

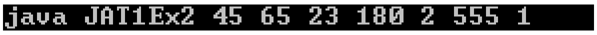
Save your class as **JAT1Ex1.java**



```
Hello World
```

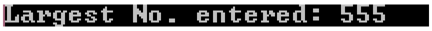
Q2 – JAT1Ex2

Write a Java program, which determines the largest number entered by a user at the command prompt. Investigate the methods of the Integer class when writing your solution.



```
java JAT1Ex2 45 65 23 180 2 555 1
```

Make use of the static imports feature when displaying the largest number in the console.

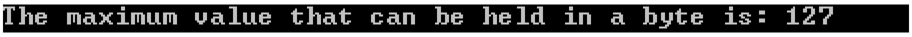


```
Largest No. entered: 555
```

Save your class as **JAT1Ex2.java**

Q3 – JAT1Ex3

Write a Java program, which displays the maximum value that can be stored in a byte primitive variable. Make use of the static imports feature when creating your program. Investigate the Byte Wrapper class in the Java API when writing your solution.



```
The maximum value that can be held in a byte is: 127
```

Save your class as **JAT1Ex3.java**

Please Turn Over.

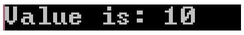
Q4 – JAT1Ex4

Please review the following classes.

```
package com.javadevelopers.projects;

public class A{
  protected int a = 10;
}
```

```
package com.javadevelopers.apps;
import com.javadevelopers.projects.*;

public class JAT1Ex4 extends A{

   public static void main(String[] args){
     new JAT1Ex4().test();
   }

   public void test(){
     System.out.println("Value is: " + new A().a);  // Compiler Error – LINE 11
   }
}
```

There is a compiler error in class JAT1Ex4 on Line 11. Make the necessary correction on Line 11 to allow the program to compile and produce the output as per the screenshot.



```
Value is: 10
```

Create a project named **JAT1Ex4**.

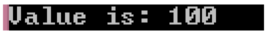
Q5 – JAT1Ex5

Please review the following classes.

```
package com.javadevelopers.projects;

public class A{
  protected int a = 100;
}
```

```
package com.javadevelopers.apps;
import com.javadevelopers.projects.*;

public class JAT1Ex5{  // Make Correction Here – LINE 4

   public static void main(String[] args){
     new JAT1Ex5().test();
   }

   public void test(){
    System.out.println("Value is: " + a);   // Compiler Error – LINE 11
   }
}
```

There is a compiler error in class JAT1Ex5 on Line 11. Make the necessary correction on Line 4 to allow the program to compile and produce the output as per the screenshot.



```
Value is: 100
```

Create a project named **JAT1Ex5**.

Please Turn Over.

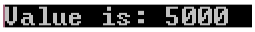
Q6 – JAT1Ex6

Please review the following classes.

```
package com.javadevelopers.projects;

public class A{
  int a = 5000;
}
```

```
package com.javadevelopers.projects;

public class JAT1Ex6{

   public static void main(String[] args){
     new JAT1Ex6().test();
   }

   public void test(){
    System.out.println("Value is: " + a);  // Compiler Error – LINE 11
   }
}
```

There is a compiler error in class JAT1Ex6 on Line 11. Make the necessary correction on Line 11 to allow the program to compile and produce the output as per the screenshot.



```
Value is: 5000
```

Create a project named **JAT1Ex6**.

Please Turn Over.

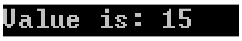
Q7 – JAT1Ex7

Please review the following classes.

```
package com.javadevelopers.projects;

public class A{
  protected int a = 15;
}
```

```
package com.javadevelopers.apps;
import com.javadevelopers.projects.*;

public class JAT1Ex7 extends A{
   public static void main(String[] args){
     new B().test();
   }
}

class B{                                              // Make Correction Here – LINE 10
 public void test(){
   System.out.println("Value is: " + a);  // Compiler Error – LINE 12
 }
}
```

There is a compiler error in class JAT1Ex7 on Line 12. Make the necessary correction on Line 10 to allow the program to compile and produce the output as per the screenshot. Class B should not be a subclass of Class A.



```
Value is: 15
```

Create a project named **JAT1Ex7**.

Please Turn Over.

Q8 – JAT1Ex8

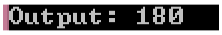
Please review the following classes.

```
package com.javadevelopers.projects;

class A{                                    // Make Correction Here – LINE 3
  public int a = 180;
}
```

```
package com.javadevelopers.apps;
import com.javadevelopers.projects.*;

public class JAT1Ex8{
   public static void main(String[] args){
     System.out.println("Output: " + new A().a);   // Compiler Error – LINE 6
   }
}
```

There is a compiler error in class JAT1Ex8 on Line 6.

Make the necessary correction on Line 3 in class A, to allow the program to compile and produce the output as per the screenshot.



```
Output: 180
```

Create a project named **JAT1Ex8**.

Q9 – JAT1Ex9

Interfaces are contracts for what a class can do.

Create an interface named, Tunable. A class which implements the Tunable interface is stating that it has implemented the functionality of the interface in a specific manner.

The interface should declare one method:

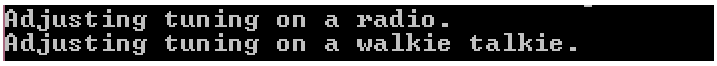
- void adjustTuning()

The following classes should implement the Tunable interface.

- Radio
  - The adjustTuning() method should display the message, “Adjusting tuning on a radio.”
- WalkieTalkie
  - The adjustTuning() method should display the message, “Adjusting tuning on a walkie talkie”.

Create a class named Main to contain the main method. Create an object from the Radio and WalkieTalkie classes respectively, and call the adjustTuning() method in turn.

Store all files in a project named **JAT1Ex9**.



```
Adjusting tuning on a radio.
Adjusting tuning on a walkie talkie.
```

Please Turn Over.

Q10 – JAT1Ex10

Create an interface named **Inflatable**.

Constants
float maxSetting = 100.00f;
float inflateFactor = 10.00f;

Abstract method
void inflate();

Create a second interface named **Deflatable**.

Constants
float minSetting = 0.00f;
float deflateFactor = 10.00f;

Abstract method
void deflate();

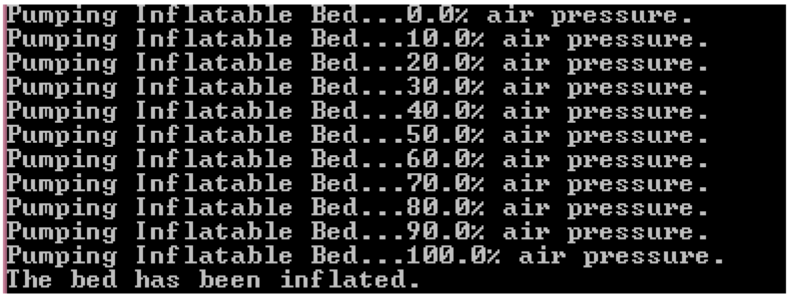
Create a class named **InflatableBed**, which implements the methods of both interfaces.

To inflate the bed, you pump air into it. The class should therefore have a variable named airAdded (a float) to measure the amount of air added.

To deflate the bed, you release air from it.

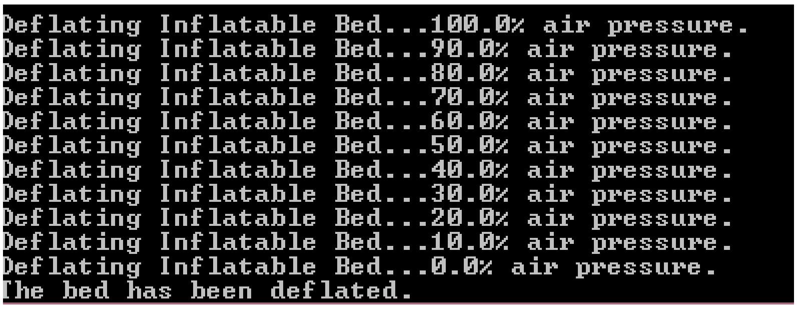
In the overridden *inflate()* method, use a looping construct of your choice to simulate the act of inflating the bed (adding air).

Make use of the *Inflatable* interface constants, *maxSetting* and *inflateFactor* in your looping construct.



```
Pumping Inflatable Bed...0.0% air pressure.
Pumping Inflatable Bed...10.0% air pressure.
Pumping Inflatable Bed...20.0% air pressure.
Pumping Inflatable Bed...30.0% air pressure.
Pumping Inflatable Bed...40.0% air pressure.
Pumping Inflatable Bed...50.0% air pressure.
Pumping Inflatable Bed...60.0% air pressure.
Pumping Inflatable Bed...70.0% air pressure.
Pumping Inflatable Bed...80.0% air pressure.
Pumping Inflatable Bed...90.0% air pressure.
Pumping Inflatable Bed...100.0% air pressure.
The bed has been inflated.
```

The overridden *deflate()* method simulates the act of deflating the bed (reducing the air). Make use of the *Deflatable* interface constants *minSetting* and *deflateFactor* in a looping construct of your choice.

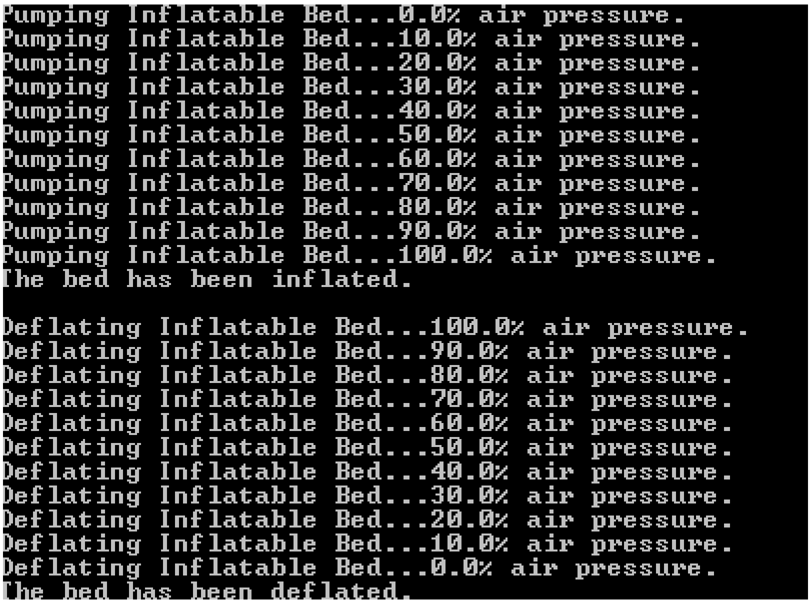


```
Deflating Inflatable Bed...100.0% air pressure.
Deflating Inflatable Bed...90.0% air pressure.
Deflating Inflatable Bed...80.0% air pressure.
Deflating Inflatable Bed...70.0% air pressure.
Deflating Inflatable Bed...60.0% air pressure.
Deflating Inflatable Bed...50.0% air pressure.
Deflating Inflatable Bed...40.0% air pressure.
Deflating Inflatable Bed...30.0% air pressure.
Deflating Inflatable Bed...20.0% air pressure.
Deflating Inflatable Bed...10.0% air pressure.
Deflating Inflatable Bed...0.0% air pressure.
The bed has been deflated.
```

Create a class named *Main* to store the main method. Create an object from the class *InflatableBed*, and call the *inflate*() and *deflate*() methods.

Store all files in a project named **JAT1Ex10**.

A full listing of the programme is shown below.



```
Pumping Inflatable Bed...0.0% air pressure.
Pumping Inflatable Bed...10.0% air pressure.
Pumping Inflatable Bed...20.0% air pressure.
Pumping Inflatable Bed...30.0% air pressure.
Pumping Inflatable Bed...40.0% air pressure.
Pumping Inflatable Bed...50.0% air pressure.
Pumping Inflatable Bed...60.0% air pressure.
Pumping Inflatable Bed...70.0% air pressure.
Pumping Inflatable Bed...80.0% air pressure.
Pumping Inflatable Bed...90.0% air pressure.
Pumping Inflatable Bed...100.0% air pressure.
The bed has been inflated.

Deflating Inflatable Bed...100.0% air pressure.
Deflating Inflatable Bed...90.0% air pressure.
Deflating Inflatable Bed...80.0% air pressure.
Deflating Inflatable Bed...70.0% air pressure.
Deflating Inflatable Bed...60.0% air pressure.
Deflating Inflatable Bed...50.0% air pressure.
Deflating Inflatable Bed...40.0% air pressure.
Deflating Inflatable Bed...30.0% air pressure.
Deflating Inflatable Bed...20.0% air pressure.
Deflating Inflatable Bed...10.0% air pressure.
Deflating Inflatable Bed...0.0% air pressure.
The bed has been deflated.
```

Please Turn Over.

Q11 – JAT1Ex11

In this assignment, you will again test your understanding of interface design and implementation.

Imagine that you are the project leader in charge of a team of software developers, who are developing a new series of phones.

As project leader, you identify the following *behaviours / functionality*, common to all phones. Each phone will implement the functionality in a specific manner.

- Make a Call
- Receive a Call
- Send a Text Message
- Receive a Text Message
- Recharge
- Hang Up

You decide to outline these common behaviours in an interface named **Communicatable**, and ask each developer who is modelling a particular type of phone to implement this interface.

The *Communicatable* interface can be seen as a contract. Every class which implements the *Communicatable* interface states that it has implemented the behaviours outlined in the interface, in a specific manner.

Methods to be outlined in the Communicatable interface.

```
public void makeCall (String noToDial);
public void receiveCall (String incomingPhoneNo);
public void sendText (String messageToSend,String noToText);
public void receiveText (String message, String incomingPhoneNo);
public void recharge(boolean status);
public void hangUp();
```

A number of smartphones in the range will also have the ability to stream live video. The implementation of this behaviour will differ depending on the model of smartphone. You decide to create a second interface named **Streamable**, which outlines the behaviour required to stream live video.

Method to be included in the Streamable interface.

```
public void streamVideo();
```

You have identified the following attributes as being standard across all types of phone. Create a class named **Phone**.

Access Modifier	**Data Type**	**Name**
private	String	name
private	int	noOfDisplayPixels
private	float	width
private	float	height
private	float	weight
private	boolean	isPoweredOn
private	boolean	isRecharging

Please Turn Over.

- A constructor should be included in the *Phone* class with seven arguments.
- Accessor methods (setter and getter methods) for the instance variables should also be declared.
- Include a toString() method in the class.

The following subclasses (of Phone) should be created to represent specific phone models.

Phone Model	**Functionality**
LandLine2000	Standard functionality*. Cannot stream live video.
G200	Standard functionality*. Can stream live video.

**Standard functionality refers to the ability to make and receive a call, send and receive a text message, recharge phone and hang-up.*

When implementing the behaviours of a particular type of phone, mention the name of the model in the narrative.

For Example:

When making a call on a LandLine2000 phone, the following message should be displayed, Dialling number <<No. to Dial>> on a LandLine2000 phone.

When receiving a call on a LandLine2000 phone, the following message should be displayed, Incoming call from <<Incoming Phone No.>> on a LandLine2000 phone.

Create a class named Main to include the main method.

In the main method, create an object from each of the following classes with the specified attributes.

- LandLine2000
- G200

LandLine2000	
name	LandLine 2000
noOfDisplayPixels	400
width	5.6
height	8.5
weight	80.5
isPoweredOn	true
isRecharging	false

G200	
name	G200
noOfDisplayPixels	510
width	4.5
height	8.6
weight	80.5
isPoweredOn	true
isRecharging	false

Create an arraylist to store the object references for the LandLine2000 and G200 phones.

Please Turn Over.

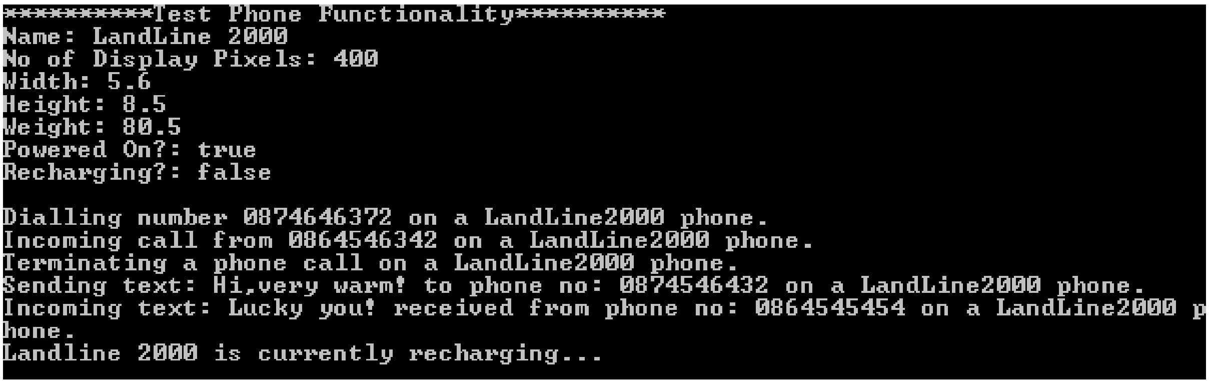
Finally, use an enhanced for loop to iterate through the arraylist. For each object reference stored, call the following methods:

- toString()
- makeCall(“0874646372”)
- receiveCall(“0864546342”)
- hangUp()
- sendText(“Hi very warm!","0874546432")
- receiveText(“Lucky you!”,0864545454”)
- recharge(true)
- streamVideo()***

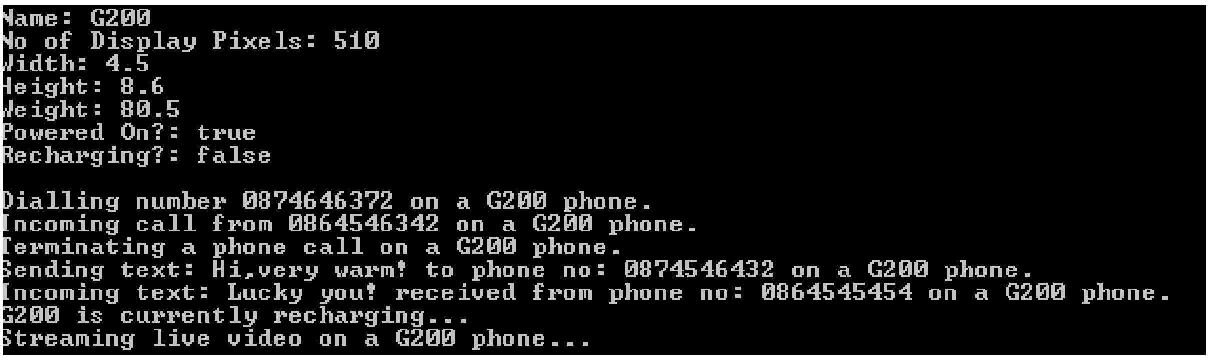
*** Note, the streamVideo() method can only be called on a G200 object reference.

Create a folder named **JAT1Ex11** to store the requested interfaces and classes.

The expected output of the program is shown below.



```
***********Test Phone Functionality***********
Name: LandLine 2000
No of Display Pixels: 400
Width: 5.6
Height: 8.5
Weight: 80.5
Powered On?: true
Recharging?: false

Dialling number 0874646372 on a LandLine2000 phone.
Incoming call from 0864546342 on a LandLine2000 phone.
Terminating a phone call on a LandLine2000 phone.
Sending text: Hi,very warm! to phone no: 0874546432 on a LandLine2000 phone.
Incoming text: Lucky you! received from phone no: 0864545454 on a LandLine2000 p
hone.
Landline 2000 is currently recharging...
```



```
Name: G200
No of Display Pixels: 510
Width: 4.5
Height: 8.6
Weight: 80.5
Powered On?: true
Recharging?: false

Dialling number 0874646372 on a G200 phone.
Incoming call from 0864546342 on a G200 phone.
Terminating a phone call on a G200 phone.
Sending text: Hi,very warm! to phone no: 0874546432 on a G200 phone.
Incoming text: Lucky you! received from phone no: 0864545454 on a G200 phone.
G200 is currently recharging...
Streaming live video on a G200 phone...
```

Please Turn Over.

Q12 – JAT1Ex12

Create the following simple class to model a Car.

```
public class Car{
  private String name;

  public Car(String name){
   this.name = name;
  }

  public void setName(){
   this.name = name;
  }

  public String getName(){
   return name;
  }
}
```

Create the following subclasses.

```
public class Honda extends Car{

 public Honda(String name){
  super("Honda");
 }
}
```

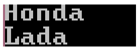
```
public class Lada extends Car{

  public Lada(String name){
   super("Lada");
  }
}
```

Create a class named JAT1Ex12 to contain the main method.

In the main method, create an array named *myCars*, to store object references of type Car or its subclasses.

Create an object of type Honda and another of type Lada. Store the object references in the array.

Finally, use an enhanced for loop to iterate through the array and call the getName() method of each object reference in turn.



```
Honda
Lada
```

Create a folder named **JAT1Ex12** to store the requested classes.

Please Turn Over.

Q13 – JAT1Ex13

In this exercise, you will test your understanding of static variables and methods.

Create a class to model the basic workings of a current account (a bank account).

Create a class named *CurrentAccount*. It should contain the following instance variables.

```
private String firstName;
private String lastName;
private String password;
private float balance;
```

- The class should contain one constructor with three parameters.
  - String firstName
  - String lastName
  - String password

- Accessor / Mutator methods should be created for each of the instance variables.
  - Note: only an accessor method should be set for the balance instance variable.

- The class should contain a method named, makeLodgement(), which has one parameter, a float to store the requested lodgement amount. The method should top up a customer's balance with the specified lodgement amount.

- The class should contain a method named makeWithdrawal(), which has one parameter, a float to store the requested withdrawal amount. If the withdrawal amount is greater than the customer's balance, the message, "Insufficient Funds", should be displayed in the console. Otherwise the customer's balance should be reduced accordingly.

- A toString() method should also be included in the class to capture the state of the instance variables of a particular object.

- Create a second class named **JAT1Ex13** to contain the main method.

- Create three objects using the details shown in the table below.

First Name	**Last Name**	**Password**
Billy	Bonds	3434S
Clare	Taylor	5441S
Anna	Long	6431S

- The following lodgements should be made.

First Name	**Last Name**	**Lodgement**
Billy	Bonds	40
Clare	Taylor	100
Anna	Long	135

Please Turn Over.

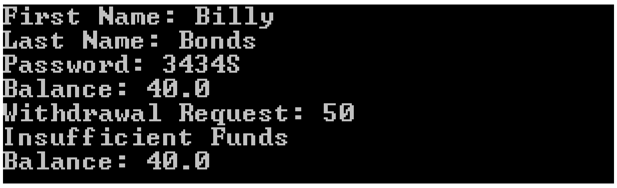
- Call the toString() method on the first object reference (Billy Bonds). Try and withdraw €50. Check the balance on the account.
- Call the toString() method on your second object reference. Try and withdraw €600. Check the balance on the account.
- Call the toString() method on your third object reference. Try and withdraw €60. Check the balance on the account.

The branch manager would like to know the following information.

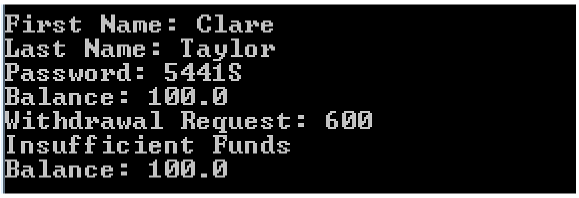
- The number of current accounts created.
- The sum of the balances held across all accounts.
- The average balance held per customer.

This information should be determined by the program using static variables and methods.

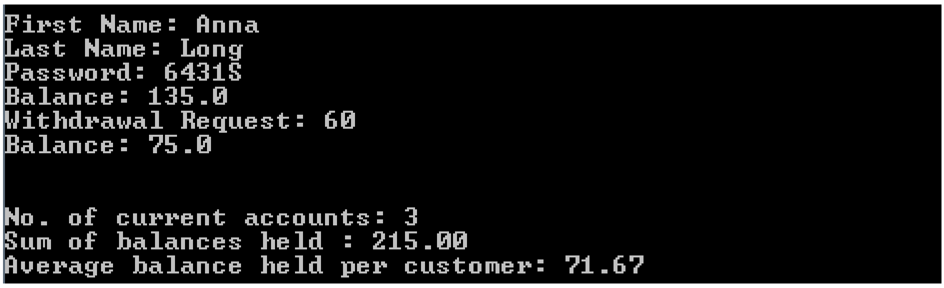
The expected output of the program is shown below.



```
First Name: Billy
Last Name: Bonds
Password: 3434S
Balance: 40.0
Withdrawal Request: 50
Insufficient Funds
Balance: 40.0
```



```
First Name: Clare
Last Name: Taylor
Password: 5441S
Balance: 100.0
Withdrawal Request: 600
Insufficient Funds
Balance: 100.0
```



```
First Name: Anna
Last Name: Long
Password: 6431S
Balance: 135.0
Withdrawal Request: 60
Balance: 75.0


No. of current accounts: 3
Sum of balances held : 215.00
Average balance held per customer: 71.67
```

`

Create a folder named **JAT1Ex13** to store the requested classes.

Please Turn Over.

Q14 – JAT1Ex14

An enum specifies a list of constant values assigned to a type.

Create a class named **JAT1Ex14** to contain the main method.

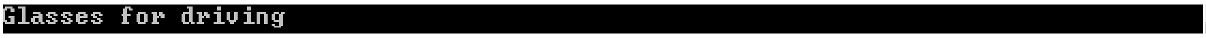
Create an enum named *Glasses* to store the following types of glasses:

- SUN
- DRIVING
- READING
- SKIING

The enum should be declared within the class. In the main method, declare a variable named myGlasses of the enum type, Glasses. Assign the value, DRIVING to the variable.

Use a switch statement to test the value stored in the myGlasses variable. Display one of the following messages.

- Glasses for sunny weather
- Glasses for driving
- Glasses for reading
- Glasses for skiing



```
Glasses for driving
```

Create a folder named **JAT1Ex14** to store the requested classes.

Q15 – JAT1Ex15

The *values*() method of an enum object returns an array of the values stored in an enum.

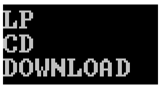
Create a source code file named, MUSIC_FORMATS.java. Within it, create an enum named MUSIC_FORMATS with public access. The following constants should be specified.

- LP
- CD
- DOWNLOAD

Create a class named **JAT1Ex15**.

In the main method, declare a variable named, **format,** of the enum type, MUSIC_FORMATS. Assign the value, CD to the variable.

Create an array to store the values held in the enum. Use an enhanced for loop to iterate through the array and display the values stored.



```
LP
CD
DOWNLOAD
```

Create a folder named **JAT1Ex15** to store the requested classes.

End of Exercises