1. **What are the differences between composition and inheritance?**

Inheritance – is when a class extends another class
Ex: class MyClass extends Superclass {...}
MyClass receives all the static and instance variables, static and instance methods of the superclass. Must be used only if a MyClass object is a Superclass object.

Composition - is when an object from a class uses an object from another class. It is a form of aggregation with strong ownership and coincident lifetimes. The parts cannot survive the whole/aggregate

2.What is polymorphism. Give a brief example.

A polymorphic variable can appear to change its type through dynamic binding.
- The compiler always understands a variable's type according to its declaration.
- The compiler permits some flexibility by way of type conformance.
- At run-time the behavior of a method call depends upon the type of the object and not the variable.

Example:

```
Base theBase;
theBase = new Doubler();
theBase = new Squarer();
theBase.printTheInt();
```

3. What is the difference between overriding and overloading. Give brief examples.

When a method is overridden, the new method definition given in the derived class has the exact same number and types of parameters as in the base class. When a method in a derived class has a different signature from the method in the base class, that is overloading.
For overloading you have the same name but different parameters so the compiler knows which method to choose.

4. Given classes A, B, and C, where B extends A, and C extends B, and where all classes implement the instance method void doIt() . How can the doIt() method in A be called from an instance method in C? Why?

You need to have a reference to the class A and the call the method using nameOfReferece.doIt().

This is because you can't call super as many times as you want. You can only do it once. Therefore you only have access to the method in class B

5.What would be the result of compiling and running the following program?

```
// Filename: MyClass. java
public class MyClass {
  public static void main(String[] args) {
    C c = new C();
    System.out.println(c. max(13, 29));
  }
}
class A {
  int max(int x, int y) { if (x>y) return x; else return y; }
}
class B extends A{
  int max(int x, int y) { return super.max(y, x) - 10; }
}
class C extends B {
  int max(int x, int y) { return super.max(x+10, y+10); }
}
```

Result: The program is compiled with no errors and the program prints on the screen: 29

6. Which is the simplest expression that can be inserted at (1), so that the program prints the value of the text field from the Message class?

```
// Filename: MyClass.java
class Message {
   // The message that should be printed:
  String text = "Hello, world! ";
}
class MySuperclass {
  Message msg = new Message() ;
}
public class MyClass extends MySuperclass {
  public static void main(String[] args) {
    MyClass object = new MyClass();
    object.print();
  }
  public void print() {
    System.out.println( /* (1) WRITE THIS COMPLETED STATEMENT */ ) ;
  }
}
```

Statement should be changed with: msg.text

7. Which method declarations, when inserted at (7), will not result in a compile-time error?

```
class MySuperclass {
  public Integer step1(int i) { return 1; } // (1)
  protected String  step2(String str1, String str2) {return str1;}  // (2)
  public String step2(String str1) { return str1; }  // (3)
  public static String step2() { return "Hi "; }  // (4)
  public MyClass makeIt() { return new MyClass(); } // (5)
  public MySuperclass makeIt2() { return new MyClass(); } // (6)
}
public class MyClass extends MySuperclass {
```

```
  // (7) WRITE THIS METHOD DECLARATION
}
        public Integer step1(int i)     { return super.step1(i); }
        protected String  step2(String str1, String str2) { return super.step2(str1); }
        public String step2(String str1)    { return super.step2(str1); }
        public static String step2()        { return "hi"; }
        public MyClass      makeIt() { return super.makeIt(); }
        public MySuperclass makeIt2() { return super.makeIt2();
```

None will result in a compile-time error.

8. What would be the result of compiling and running the following program?

```
class Vehicle {
  static public String getModelName() { return "Volvo"; }
  public long getRegNo() { return 12345; }
}
class Car extends Vehicle {
  static public String getModelName() { return "Toyota"; }
  public long getRegNo() { return 54321; }
}
public class TakeARide {
  public static void main(String args[]) {
    Car c = new Car();
    Vehicle v = c;
    System.out.printl n("| " + v.getModelName() + "| " + c.getModel Name() +
  "| " + v.getRegNo() + "| " + c.getRegNo() + "| ");
  }
}
```

Output: | Volvo| Toyota| 54321| 54321|

9.What would be the result of compiling and running the following program?

```
final class Item {
  Integer size;
  Item(Integer size) { this. size = size; }
  public boolean equals(Item item2) {
    if (this == item2) return true;
    return this.size.equals(item2.size);
  }
}
public class SkepticRide {
  public static void mai n(String[] args) {
    Item itemA = new Item(10);
    Item itemB = new Item(10);
    Object itemC = itemA;
    System.out.printl n("| " + itemA.equals(itemB) +
      "| " + itemC.equals(itemB) + "| ") ;
  }
}
```

Output: | true| false|

10.Which constructors can be inserted at (1) in MySub without causing a compile-time error?

```
class MySuper {
  int number;
  MySuper(int i) { number = i ; }
}
class MySub extends MySuper {
  int count;
  MySub(int count, int num) {
    super(num);
    this.count = count;
  }
  // (1) WRITE CONSTRUCTOR NEEDED At THIS POINT
MySub(int count, int num) {
     super.number=num;
     this.count=count;
}
}
```

11.What will the following program print when run?

```
// Filename: MyClass.java
public class MyClass {
  public static void main(String[] args) {
    B b = new B("Test");
  }
}
class A {
  A() { this("1", "2"); }
  A(String s, String t) { this(s + t); }
  A(String s) { System.out.println(s); }
}
class B extends A {
  B(String s) { System.out.println(s); }
  B(String s, String t) { this(t + s + "3"); }
  B() { super("4"); };
}
```

Output: 12
Test

12. Consider the following two class definitions.

```
class X {
  public double g(double x) {
    return f(x) * f(x);
  }
  public double f(double x) {
    return x + 1.0;
  }
}
```

```
class Y extends X {
  public double f(double x) {
    return x + 2.0;
  }
}
```

What will the following sequence of statements print?

```
Y y = new Y();
X x = y;
System.out.println(y.f(2.0));
System.out.println(x.f(2.0));
System.out.println(x.g(2.0));
```

Output: 4.0
4.0
16.0

13. Suppose a user executes the following run() method and types the numbers 5 and 22.

```
public static void run() {
    int a = IO.readInt();
    int b = IO.readInt();

    int m = 0;
    int n = b;
    while(m < n) {
        m += a;
        n -= a;
    }
    System.out.println(m - n);
}
```

Show all the values taken on by the variables of the program.

a=5
b=22
m=0,5,10,15
n=22,17,12,7

What does the method print? 8

14. Complete the below class method so that it reads two strings from the user and displays "yes"' if the second string includes only characters from the first, and "no"' if it contains any characters that the first does not include. For example, I should be able to execute your method and see the following. (Boldface indicates what the user types.)

brillig glib
yes

Or I might see the following. In this example, it prints "no" because broil includes the letter o, which does not occur in brillig.

brillig broil
no

```
public static void run() {

    String a = IO.readString();
    String b = IO.readString();
    boolean ok=true;
    for (int i=0;i<length(a);i++)
    {
            boolean ok1=false;
            for (j=0;j<length(b);j++)
            {
                    if (a.charAt(i)==b.charAt(j))   {  ok1=true; }
            }
            if (ok1==false)
            {
                    ok=false;
            }
    }
    if (ok==true) { System.out.println("yes");} else  { System.out.println("no");}
}
```

To accomplish this, you may find the following String instance methods useful.

int length()

Returns the number of characters in the target string.

char charAt(int i)

Returns the character at index i. For example, if str holds the string brillig, str.charAt(2) would return the character 'i'.

int indexOf(String s)

Returns the index where s occurs first within the string on which the method is called. If s occurs nowhere within the target string, the method returns -1. For example, if str holds the string brillig, str.indexOf("il") would return 2, since this is the index where il occurs first.

15. At right, write a definition of a new type, called IntRange, representing a range of integers. This class should support the following instance methods.

IntRange(int start, int end)

(Constructor method) Sets up a new IntRange object representing the set of integers between start and end, including these two endpoints. The method may assume that the first parameter is below or equal to the second parameter.

int getSize()

Returns the number of integers in the range.

boolean contains(int i)

Returns true if i lies within the range.

The following example method, which uses the IntRange class you define, illustrates how the class should work.

```
public static void run() {
       IntRange range = new IntRange(2, 4);
       System.out.println(range.getSize());         // this prints ``3"
       System.out.println(range.contains(1));       // this prints ``false"
        System.out.println(range.contains(3));      // this prints ``true"
       System.out.println(range.contains(4));       // this prints ``true"
       System.out.println(range.contains(5));       // this prints ``false"
     }
public class IntRange{

   private int st=0,end=0;
   private int[] arr=new int[end-st+1];

   public IntRange(int start,int end)
   {
      this.st=start;
      this.end=end;
      arr=new int[end-st+1];
      int j=0;
      for (int i=st;i<=end;i++)
           arr[j]=i;
           j++;
      }
   public int getSize()
   {
     return end-st+1;//or arr.length
   }

   public boolean contains(int i)
   {
      if(i>=st&&i<=end)
        return true;
      else
         return false;

}
}
```

16. Suppose we have the following two class definitions.

```
class A {                  class B extends A {
  public int f(int x) {      public int f(int x) {
    return 2 * x;              return 5 * x;
  }                          }
  public int g(int x) {      public int h(int x) {
    return f(x * 3);           return f(7 * x);
  }                          }
}                          }
```

And suppose we execute the following run method.

```
public static void run() {
  B b = new B();
  System.out.println(b.f(1) + " " + b.g(1) + " " + b.h(1));
  A a = b;
```

```
  System.out.println(a.f(1) + " " + a.g(1));
}
```

What would the method print?

Output:
```
5 15 35
5 15
```

17. Define a class PassCount to track whether all the students of a class have passed a test. It should support the following methods.

PassCount()
(Constructor method) Constructs an object representing an empty class.
void addGrade(double grade) Adds grade into the class.
boolean isAnyFailing() Returns true only if there is somebody in the class who received a grade of less than 60.
For example, if you defined this class properly, I should be able to write the following class to test it.

```
public class PassCountTest {
    public static void run() {
    PassCount a = new PassCount();
    addGrade(45.0);
    addGrade(76.0);
    System.out.println(a.isAnyFailing()); // should print "true"

    PassCount b = new PassCount();
    addGrade(60.0);
    System.out.println(b.isAnyFailing()); // should print "false"
    }
}
```

Note that, to accomplish this, a PassCount object need not remember every single grade --- that is, you do not need to use arrays: It just needs to remember whether any of the ones it has seen were below 60.

```
public PassCount{
public double g;
   PassCount()
   {this.g=g;  }

   public void addGrade(double grade)
   {
      if(grade<(60.0))
         g=grade;
   }
  public boolean isAnyFailing()
   {
      if(g<(60.0))
         return true;
     else    return false;
    }
```

18. Define a class NumberIterator for iterating through a sequence of numbers. It should support the following methods.

NumberIterator(int start, int stop)

(Constructor method) Constructs an object for iterating through the integers beginning at start and going up to stop. The constructor assumes that start is less than stop.

boolean hasMoreNumbers()

Returns true if there are more numbers remaining in the sequence.

int nextNumber()

Returns the current number in the sequence and steps the iterator forward, so that the next call to this method returns the following number in the sequence. This method initiates a NoSuchElementException if the sequence has no more elements remaining.

For example, if you defined this class properly, I should be able to write the following class to test it. When executed, its run() method would print "5 6 7 8".

```
public class NumberIteratorTest {
   public static void run() {
      NumberIterator it = new NumberIterator(5, 8);
      System.out.print(it.nextNumber());
      while(it.hasMoreNumbers()) {
         int i = it.nextNumber();
         System.out.print(" " + i);
      }
   }
}
```

```
public class NumberIterator {
public int[] number =new int[1000];
public int s;
public int st;
public int it=0;
public NumberIterator(int start,int stop)
{for(int i=start;i<=stop;i++)
{it++;
number[it]=i;
}
s=start;
st=stop;
it=0;
}
boolean hasMoreNumbers(){
if (it<st-s+1)
return true;
else
return false;
}
int nextNumber(){
if (it<st-s+1)
{it++;
return number[it];

}else
throw new NoSuchElementException
return -1;
}}
```

19. Suppose we have the following two class definitions.

```
class P {
  public int f(int x) {
    return x + 1;
  }
  public int g(int x) {
    return f(x + 2);
  }
}
```

```
class Q extends P {
  public int f(int x) {
    return x + 4;
  }
  public int h(int x) {
    return f(x + 8);
  }
}
```

And suppose we execute the following run method.

```
public static void run() {
  P a = new P();
  Q b = new Q();
  P c = b;
  System.out.println(a.f(0) + " " + a.g(0));
  System.out.println(b.f(0) + " " + b.g(0) + " " + b.h(0));
  System.out.println(c.f(0) + " " + c.g(0));
}
```

What would the method print?

Output:
```
1  3
4  6  12
4  6
```

20. Suppose we have the class defined as below.

```
class C {
   static int y = 0;
   int z;

   C() {
      z = 0;
   }
   void incrX() {
      int x = 0;
      x++;
      IO.println(x);
   }
   void incrY() {
      y++;
      IO.println(y);
   }
   void incrZ() {
      z++;
      IO.println(z);
   }
}
```

```
public class Example {
   public static void run() {
      C a = new C();
      C b = new C();
      a.incrX();
      a.incrX();
      b.incrX();
      a.incrY();
      a.incrY();
      b.incrY();
      a.incrZ();
      a.incrZ();
      b.incrZ();
   }
}
```

What would the computer print when it executes the Example class's run() method?

Output: 1 1 1 1 2 3 1 2 1

21. A matrix is symmetric if, for each i and j, $a_{i,j} = a_{j,i}$. The following is an example of a symmetric matrix.

0 23 45
23 10 36
46 36 20

Another way of defining it: A symmetric matrix can be reflected across its main diagonal (top left corner to bottom right corner) to obtain the same matrix.

Complete the following method so that it returns true only if the two-dimensional array parameter mat is symmetric. Your solution may assume that the matrix is square (as many columns as rows).

```
public static boolean isSymmetric(int[][] mat) {
Boolean ok = true;
n=mat.length;
for (int i=0;i<n;i++){
        for (int j=0;j<n;j++){
                if (i<>j) {
                        if a[i][j]<>a[j][i] { ok=false;}
                }
        }
}
return ok;
}
```

(Your solution shouldn't really be this long!)

22.Define a class PigPen for tracking the number of pigs in a pen. It should support the following methods.

PigPen(int pigs)
(Constructor method) Constructs an object representing a pig pen containing pigs pigs.
boolean isEmpty()
Returns true if there are no pigs in the pen.
void pigEnters()
Adds one to the number of pigs in the pen.
void pigExits()
Subtracts one from the number of pigs in the pen.
For example, if you defined this class properly, I should be able to write the following class to test it.

```
public class PigPenTest {
   public static void run() {
      PigPen pen = new PigPen(2);
      pen.pigExits();
      System.out.println(pen.isEmpty()); // prints "false"
      pen.pigExits();
      System.out.println(pen.isEmpty()); // prints "true"
      pen.pigEnters();
      System.out.println(pen.isEmpty()); // prints "false"
   }
}
```

```
public class PigPen {
   int pigs = 0;
   PigPen(int pigs) {
      this.pigs = pigs;
   }
   boolean isEmpty() {
      if (this.pigs == 0) return true;
      else return false;
   }
   void pigEnters() {
      this.pigs++;
   }
   void pigExits() {
      this.pigs--;
   } }
```

23. Write a class method named mode that takes an array of ints as a parameter and returns the integer that occurs in the array most frequently. For example, the following code fragment that uses your mode method should print 23.

```
int[] a = { 23, 34, 45, 23, 0, 23 };
System.out.println(mode(a));
```

Your method should not call any other methods to accomplish this. It will need more than one loop to count the number of occurrences of each number in the array.

```
public int mode(int[] a)
   {
      int n=a.length;
      int[] aux=new int[a.length];
      for(int i=0;i<n;i++)
            aux[i]=0;
     for( int i=0;i<n;i++)
      {
         for (int j=0;j<i-1;j++)
         {if (a[i]==a[j])
             aux[i]=aux[i]+1;
         }
      }
      int max=0;
      for(int i=0;i<n;i++)
           if (aux[i]>max)
             max=aux[i];
      return max+1;
   }
```

24.Consider the following Java program.

```
class A {
   int f() { return 1; }
}
class B extends A {
   int f() { return 0; }
}
class Main {
   public static void main(String[] args) {
      A a = new B();
      System.out.println(a.f());
   }
}
```

What does this program print?

Output: 0

25.Consider the following program.

```
class Ident {
   int f(int x) { return x; }
   int g(int x) { return f(f(x)); }
}
class Square extends Ident {
```

```
    int f(int x) { return x * x; }
}
class Main {
    public static void main(String[] args) {
        Ident a = new Ident();
        Ident b = new Square();
        Square c = new Square();
        System.out.println(a.g(3) + " " + b.g(3) + " " + c.g(3));
    }
}
```

What does this program print?
Output: 3 81 81

26.Write an expression that will extract the substring "kap", given the following declaration:

```
String str = "kakapo";
```

```
String str1 = str.substring(2,5);
```

27. What will be the result of attempting to compile and run the following code?

```
class MyClass {
  public static void main(String[] args) {
    String str1 = "str1";
    String str2 = "str2";
    String str3 = "str3";
    str1.concat(str2);
    System.out.println(str3.concat(str1));
  }
}
```

Output: str3str1

28. What will be the result of attempting to compile and run the following program?

```
public class RefEq {
  public static void main(String[] args) {
    String s = "ab" + "12";
    String t = "ab" + 12;
    String u = new String("ab12");
    System.out.println((s==t) + " " + (s==u));
  }
}
```

Output: true false

29. What is a Java exception?

An exception: a problem that occurs when a program is running.

--When an exception occurs, the JVM creates an object of class **Exception** which holds information about the problem.

--A Java program itself may catch an exception. It can then use the Exception object to recover from the problem.

30.How can one deal with exceptional conditions in Java. Give a short example.

Java library software (or programmer-defined code) provides a mechanism that signals when something unusual happens. This is called throwing an exception. In another place in the program, the
programmer must provide code that deals with the exceptional case. This is called handling the exception.

You can deal with an exception by "catching" it. This allows the program to continue working even though something unusual has happened. Catching is done in a try-throw-catch mechanism but the catch and try don't have to be in the same class.

EX: you attempt to access an array at index 10 but the array is of length 5. You get an arrayOutOfBounds Exception which you can catch in the catch statement and tell the program what to do if this happens. For example you can access the last element of the array.

31.Describe the try-throw-catch mechanism.

The **try** block contains the code for the basic algorithm. It tells what to do when everything goes smoothly. It is called a try block because it "tries" to execute the case where all goes as planned

When an exception is thrown, the execution of the surrounding **try** block is stopped. The value thrown is the argument to the **throw** operator, and is always an object of some exception class.

throw statement is similar to a method call:

throw new *ExceptionClassName*(*SomeString*);

Instead of calling a method, a **throw** statement calls a **catch** block. When an exception is thrown, the **catch** block begins execution. The **catch** block has one parameter

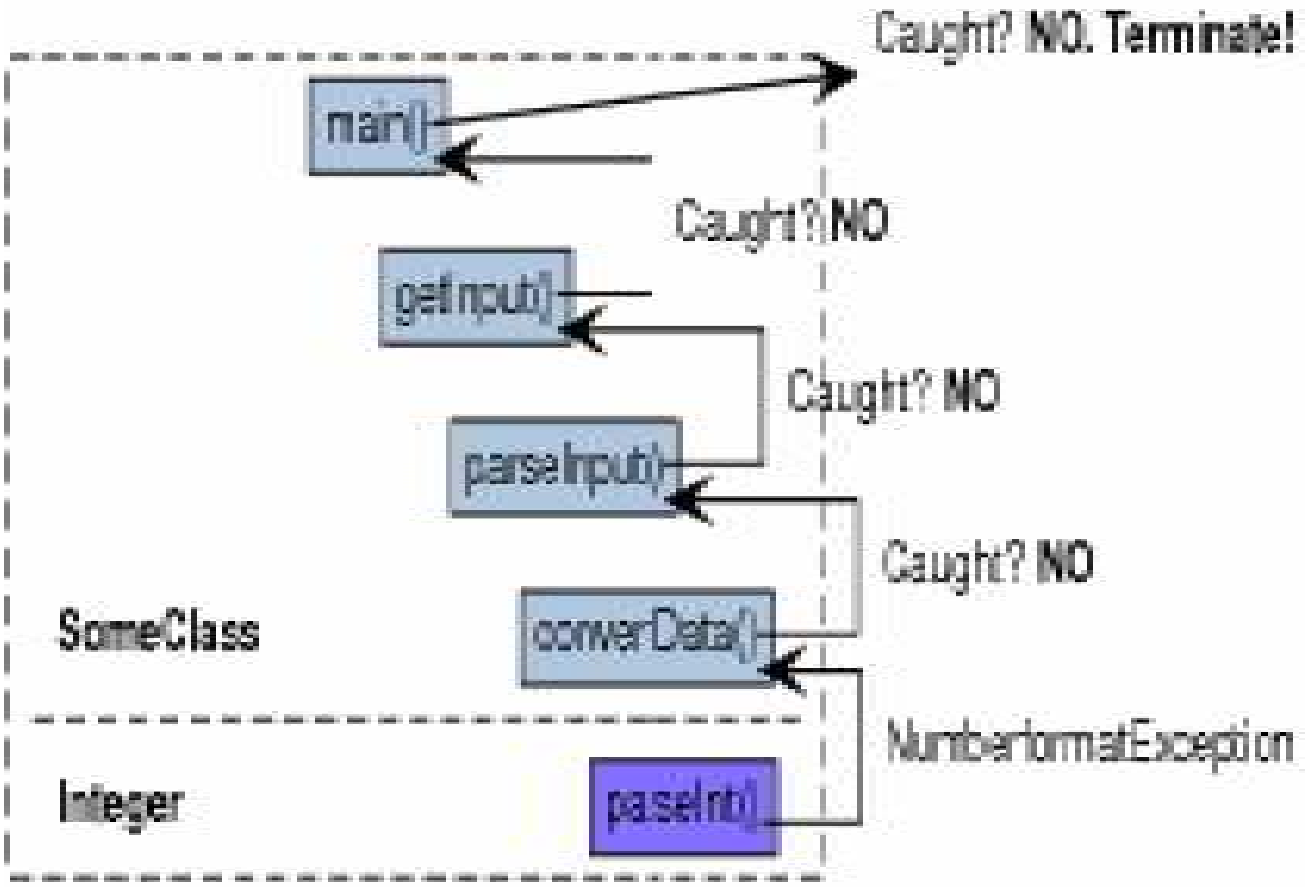
The execution of the **catch** block is called "catching the exception", or handling the exception

32.State the catch-or-declare rule.

Most ordinary exceptions that might be thrown within a method must be accounted for in one of two ways:

1. The code that can throw an exception is placed within a **try** block, and the possible exception is caught in a **catch** block within the same method
2. The possible exception can be declared at the start of the method definition by placing the exception class name in a **throws** clause

33.How do exceptions propagate? Give a brief example.



EX: put a catch statement in parseInt() and there the NumberFormatException will be dealth with.

34.What is the minimum a user-defined exception class must contain? Give a short example.

Every exception class to be defined must be a derived class of some already defined exception class. It can be a derived class of any exception class in the standard Java libraries, or of any programmer defined exception class.

Constructors are the most important members to define in an exception class. They must behave appropriately with respect to the variables and methods inherited from the base class

```
public class DivisionByZeroException extends Exception
{
    public DivisionByZeroException()
    {
            super("Division by zero.");
    }
    public DivisionByZeroException(String message)
    {
    super(message);
    }
}
```

35.What is an *inner* class?

A class definition within another class definition is called an inner class

- it allows you to group classes that logically belong together and to control the visibility of one within the other
- inner classes are distinctly different from composition

36.What is an *anonymous inner* class? Give a short example.

An anonymous inner class is a local class that has no name. They cannot have explicit constructors

```
class MyOuter {
    class  {
   //...
   public void function1() {}
   //more functions
   }
}
```

```
public class DirList2 {
  public static FilenameFilter filter(final String regex) {
    // Creation of anonymous inner class:
    return new FilenameFilter() {
      private Pattern pattern = Pattern.compile(regex);

      public boolean accept(File dir, String name) {
        return pattern.matcher(new File(name).getName()).matches();
      }
    }; // End of anonymous inner class
  }
```

37.What is a local class? Give a short example.

Local classes are declared within a block of code and are visible only within that block, just as any other method variable.

```
interface MyInterface {
   public String getInfo();
}
class MyOuter {
   MyInterface current_object;
   public void setInterface(String info) {
         class MyInner implements MyInterface {
            private String info;
            public MyInner(String inf) {info=inf;}
            public String getInfo() {return info;}
            }
   current_object = new MyInner(info);
   }
}
```

38.What is a Java *component*? Give a short example.

A Java Component is visual object containing text or graphics, that can be displayed on the screen and can interact with the user. (Class Component declares the common attributes and behaviors of all its subclasses. A component class is actually any descendent class of the class JComponent. Any JComponent object or component can be added to any container class object (a JComponent can also be added to another JComponent))

Ex: Canvas, JButton, JTextField, JLabel

39.What is a Java *container*? Give a short example.

A container is a graphical object that can hold components or other containers. The principal container is a Frame. It is a part of the computer screen surrounded by borders and title bars.
(Class Container manages a collection of related components. Any class that is a descendent class of the class Container is considered to be a Container class (JFrame and JPanel are descendent classes => they and any of their descendents can serve as a container))

40.What are the Basic steps in displaying Java Graphics?

1. Create the component or components to display
2. Create a frame to hold the component(s), and place the component(s) into the frame(s).
3. Create a “listener” object to detect and respond to mouse clicks, and assign the listener to the frame.

Also you can use panels for the components. Put components in panels and set panels in the frame

41.What is a listener in Java?

A *listener* is called when the user does something to the user interface that causes an *event.* It belongs to a class that is provided by the application programmer. Its methods describe the actions to be taken when an event occurs. A “listener” class listens for mouse clicks or keyboard input on a container or component, and responds when it occurs

42.What is the purpose of call-backs?

Callback is a scheme used in event-driven programs where the program registers a subroutine (a "callback handler") to handle a certain event.

The program does not call the handler directly but when the event occurs, the run-time system calls the handler, usually passing it arguments to describe the event.

43.What is the role of the *model* in the MVC architecture?

The Model is the part that does the work – it models the actual problem being solved.

The Model should be independent of both the Controller and the View, but it can provide services (methods) for them to use. The Model should not depend on the Controller and View.

Independence gives flexibility, robustness.

44.What is the role of the *controller* in the MVC architecture?

The Controller decides what the model is to do. Often, the user is put in control by means of a GUI.

In this case, the GUI and the Controller are often the same

The Controller and the Model can almost always be separated (what to do versus how to do it). The design of the Controller depends on the Model. (The Model should not depend on the Controller). So a controller takes user input on the view and translates that to changes in the model. The Controller should talk to the Model and View, not manipulate them (it can set variables that the Model and View can read).

45.What is the role of the *view* in the MVC architecture?

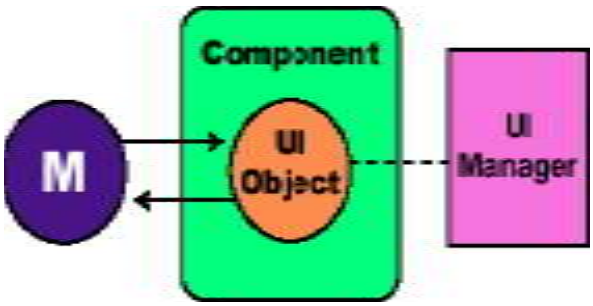
Typically, the user has to be able to see, or view, what the program is doing.

The View shows what the Model is doing. The View is a passive observer; it should not affect the model. The View should not display what the Controller thinks is happening.

The Model should be independent of the View, but (but it can provide access methods). The View should not display what the Controller thinks is happening

46.Briefly describe the Swing separable model architecture.

The view and controller parts of a component required a tight coupling. These two entities were collapsed into a single UI (userinterface) object. This new quasi-MVC design is sometimes referred to a
separable model architecture.



47.What is a Java Enumeration? Give a small example.

A simple definition of an enumeration would be: a set of constants to represent various values. An enum is actually a new type of class, it can be declared as an inner or outer class.

They are objects used to step through a container.

Available for some standard container classes which implement the corresponding interfaces.

Work properly even if the container changes.

Order might or might not be significant.

Ex: enum Season {WINTER, SPRING, SUMMER, FALL}

(To get an Enumeration **e** for container **v**:

```
Enumeration e = v.elements(); //This is initialized at the start of the list.
//To get the first and subsequent elements:
someObject = e.nextElement()
//To test if all have been accessed:
e.hasMoreElements()
```

Example:

```
  for (Enumeration e = v.elements();
  e.hasMoreElements(); ) {
  System.out.println(
  e.nextElement());

  )
```

48.What is a *generic type* in Java? Give a small example.

A generic type is defined in terms of some other type which it collects or on which it acts in some way, using angle brackets (< >)

Example: an ArrayList<Point> is an array-list of Point objects (from the java.awt package)
ArrayList<Point> someList = new ArrayList<Point>();

Allows the compiler to catch a mistake like:

somelist.add(new Dimension(10, 10));

A type-specific collection object will provided a type specific iterator

Iterator<Point> each = someList.iterator();

Then it is no longer necessary to downcast the results of accessor methods, e.g.

```
while (each.hasNext())
each.next().x = 11;
```

49.Write a generic method which prints the values stored in a collection.

```
public class Box<T> {
private List<T> contents;
public Box() {
contents = new ArrayList<T>();
}
public void add(T thing) { contents.add(thing); }
public T grab() {
if (contents.size() > 0) return contents.remove(0);
else return null;
}
```

50.What is an *iterator* in Java? Give a small example.

An iterator is an object that is used with a collection to provide sequential access to the collection elements. This access allows examination and possible modification of the elements

An iterator imposes an ordering on the elements of a collection even if the collection itself does not impose any order on the elements it contains

If the collection does impose an ordering on its elements, then the iterator will use the same ordering

51.How can one include primitive values in a Java Collection? Give a short example.

You cannot include primitive values in collections although there is a way to box them.
An example is given: Integer integerYear = new Integer(1989); Instead of 1989 you can put an int variable

52.What is a Java Collection?

A Java collection: any class that holds objects and implements the **Collection** interface. The **Collection** interface is the highest level of Java's framework for collection classes. Collections are used along with iterators
(For example, the ArrayList<T> class is a java collection class and implements all the methods in the Collection interface.)

53.What are the main differences between classes Vector and ArrayList?

For most purposes, the **ArrayList<T>** and **Vector<T>** are equivalent.

- The **Vector<T>** class is older, and had to be retrofitted with extra method names to make it fit into the collection framework
- The **ArrayList<T>** class is newer, and was created as part of the Java collection framework
- The **ArrayList<T>** class is supposedly more efficient than the **Vector<T>** class also

54.What is the purpose of software testing?

Software testing: the process used to help identify the correctness, completeness, security and quality of developed computer software.

55.What is functional testing?

Goal functional testing: determine system meets customer's specifications.

56.What defines a test case?

A test case is defined by

- Statement of case objectives;
- Data set for the case;
- Expected results.

57.What must be considered when developing a test plan?

A test plan is a set of test cases. To develop it:

-Analyze feature to identify test cases.
-Consider set of possible states object can assume.
-Tests must be representative.

58.What is a unit test?

The single most important testing tool

- Checks a single method or a set of cooperating methods
- You don't test the complete program that you are developing; you test the classes in isolation
- For each test, you provide a simple class called a test harness
- Test harness feeds parameters to the methods being tested

59.What is a test harness?

For each test in the unit test, you provide a simple class called a test harness. Test harness feeds parameters to the methods being tested. Writing test harnesses is a good solution in cases when repeating unit tests is necessary.

60.What is regression testing?

Regression testing: repeating previous tests to ensure that known failures of prior versions do not appear in new versions

61.What is test coverage?

Test coverage: measure of how many parts of a program have been tested. Makes sure that each part of your program is exercised at least once by one test case

62.How can one get a program trace? (Hint: there are at least 2 ways)

- Use the **Logger** class to turn off the trace messages without removing them from the program (**java.util.logging**)

- Debugging. A debugger lets you stop and restart your program, see contents of variables, and step through it

63.What are the benefits of logging?

Logging can generate detailed information about the operation of an application. Once added to an application, logging requires no human intervention.

- Application logs can be saved and studied at a later time.
- If sufficiently detailed and properly formatted, application logs can provide audit trails.
- By capturing errors that may not be reported to users, logging can help support staff with troubleshooting.
- By capturing very detailed and programmer-specified messages, logging can help programmers with debugging.
- Logging can be a debugging tool where debuggers are not available, which is often the case with multi-threaded or distributed applications.
- Logging stays with the application and can be used anytime the application is run.

64.What are the shortcomings of logging?

- Logging adds runtime overhead, from generating log messages and from device I/O.
- Logging adds programming overhead, because extra code has to be written to generate the log messages.
- Logging increases the size of code.
- If logs are too verbose or badly formatted, extracting information from them can be difficult.
- Logging statements can decrease the legibility of code.
- If log messages are not maintained with the surrounding code, they can cause confusion and can become a maintenance issue.
- If not added during initial development, adding logging can require a lot of work modifying code.

65.What will the following program print when run?

```
public class UppTurn {
  public static void main(String[] args) {
    String str1 = "lower", str2 = "LOWER", str3 = "UPPER";
    str1.toUpperCase();
    str1.replace("LOWER", "UPPER");
    System.out.println((str1. equals(str2)) + " " + (str1. equals(str3))) ;
  }
}
```

Output: false false

66.Describe the steps needed to read strings from a formatted sequential file.

- Select a **FileReader** class to read formatted sequential data.
- Open the file by creating a **FileReader** object
- Wrap the **FileReader** in a **BufferedReader** for efficiency
- Read the file with the **BufferedReader** method **readLine()**.
- Close file with the **FileReader** method **close()**.
- Handle I/O exceptions with a **try/catch** structure.

67. What will the method length() in the class File return?

Returns the length in bytes of the file defined in its abstract path, or 0 if the file doesn't exist

68. If write(0x01234567) is called on an instance of OutputStream, what will be written to the destination of the stream?

It writes 135 at the outputstream (only the first 8 bits from the number's representation in binary). The representation is: 0000 0000 0001 0010 1101 0110 *1000 0111* only the last 8 remain.The rest of 24 bits are ignored

69.Given the following program:

```
import java.io.DataInputStream;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
public class Endings {
  public static void main(String[] args) {
    try {
      FileInputStream fos = new FileInputStream("info.dat") ;
      DataInputStream dis = new DataInputStream(fos);
      int i= dis.readByte();
      while (i != -1) {
        System.out.print((byte)i + "| ");
        i= dis.readByte();
      }
    } catch (FileNotFoundException fnf) {
      System.out.println("File not found");
    } catch (EOFException eofe) {
      System.out.println("End of stream");
    } catch (IOException ioe) {
      System.out.println("Input error");
    }
  }
}
```

What will happen when one attempts to compile and run it?

If file info.dat doesn't exist: "File not found", if it exists: will be printed the ASCII codes of the characters contained in info.dat, followed by: "End of stream".

70.How many methods are defined in the Serializable interface?

Serializable interface has no methods.

71.Given the following code:

```
public class Person  {
  protected String name;
  Person() { }
  Person(String name) { this. name = name; }
}
```

```
import java.io. Serializable;
public class Student extends Person implements Serializable {
  private long studNum;
  Student(String name, long studNum) {
    super(name);
    this. studNum = studNum;
  }
  public String toString() { return "(" + name + ", " + studNum + ") "; }
}
```

```
import java.io.*;
public class RQ800_10 {
public static void main(String args[])
    throws IOException, ClassNotFoundException {
    FileOutputStream outputFile = new FileOutputStream("storage. dat");
    ObjectOutputStream outputStream = new ObjectOutputStream(outputFile);
    Student stud1 = new Student("Aesop", 100);
    System.out.print(stud1);
    outputStream. writeObject(stud1);
    outputStream. flush();
    outputStream. close();
    FileInputStream inputFile = new FileInputStream("storage. dat");
    ObjectInputStream inputStream = new ObjectInputStream(inputFile);
    Student stud2 = (Student) inputStream.readObject();
    System.out.println(stud2);
    inputStream. close();
  }
}
```

What will happen when one attempts to compile and run it?

Output: (Aesop, 100) (null, 100)

72.What will happen when one attempts to compile and run the following code:

```
import java.util. ArrayList;
import java.util. Collections;
import java.util. List;
public class WhatIsThis {
  public static void main(String[] args) {
    List<StringBuilder> list = new ArrayList<StringBuilder>();
    list.add("B");
    list.add("A");
```

```
        list.add("C");
        Collections.sort(list, Collections.reverseOrder());
        System.out.println(list.subList(1, 2));
    }
}
```

It will generate an error: cannot find symbol-method add(java.lang.String)
Append should be used for the StringBuilder

73.How can one achieve object persistence in Java?

There are 2 ways: Serialization or using an XML Encoder

You can take help from Java IO system to store the object in the file system. However there are convenient approaches you can meet your expectations in this regard. One way is the textual representation of the object graph in the file system and another way is the binary representation of the object graph. These ways are very much convenient and easy from the view point of development. You can achieve the textual representation of the object graph using XML Encoder and you can achieve the binary representation of the object graph using java object serialization process.

74.What is a Java Buffer?

Buffer: a linear, finite sequence of elements of a specific primitive type

- Consolidate I/O operations
- Four properties (all having never negative values)
    - Capacity: number of elements it contains (never changes)
    - Limit: index of the first element that should not be read or written
    - Position: index of the next element to be read or written
    - Mark: index to which its position will be reset when the **reset()** method is invoked

Invariant: 0 <= mark <= position <= limit <= capacity

Buffers were created primarily to act as containers for data being sent to or received from channels. (Channels are conduits to low-level I/O services and are always byte-oriented; they only know how to use **ByteBuffer** objects.)

75.What are the benefits of Buffer *views*?

Assume we have a file containing Unicode characters stored as 16-bit values (UTF-16 not UTF-8 encoding. UTF = Unicode Transformation Format)

To read a chunk of this file into the byte buffer, we could then create a **CharBuffer** view of those bytes:

CharBuffer charBuffer = byteBuffer.asCharBuffer();

This creates a view of the original **ByteBuffer**, which behaves like a **CharBuffer** (combines each pair of bytes in the buffer into a 16-bit char value)

The **ByteBuffer** class also has methods to do ad hoc accesses of individual primitive values. For example, to access four bytes of a buffer as an **int**, you could do the following:

int fileSize = byteBuffer.getInt();

76.What is the purpose of a direct Buffer? Give a brief example.

In case of direct buffers the data elements encapsulated by the buffer are stored in native memory space outside of the JVM's memory heap.

The primary purpose of direct buffers is for doing I/O on channels.

Channel implementations can set up OS-level I/O operations to act directly upon a direct buffer's native memory space.

When we create a direct buffer (by invoking **ByteBuffer.allocateDirect()**), native system memory is allocated and a buffer object is wrapped around it.

Channel implementations can set up OS-level I/O operations to act directly upon a direct buffer's native memory space.

77.What is a Java Channel?

Channels are conduits to low-level I/O services and are always byte-oriented; they only know how to use **ByteBuffer** objects.

(NET: A channel represents an open connection to an entity such as a hardware device, a file, a network socket, or a program component that is capable of performing one or more distinct I/O operations, for example reading or writing.

A channel is either open or closed. A channel is open upon creation, and once closed it remains closed. Once a channel is closed, any attempt to invoke an I/O operation upon it will cause a ClosedChannelException to be thrown. Whether or not a channel is open may be tested by invoking its isOpen method.)

78.When does the execution of a thread end?

A thread stops in three ways:

- It returns smoothly from **run().**[Best way]
- The thread's **stop()** method is called. (Now deprecated. Don't use this.)
- Interrupted by an uncaught exception.

79.How can the priority of a thread be set?

The thread scheduler runs each thread for a short amount of time (a time slice)

Then the scheduler activates another thread. There will always be slight variations in running times especially when calling operating system services (e.g. input and output)

There is no guarantee about the order in which threads are executed

You can also change the priority of a thread but this is not recommended

setPriority

public final void **setPriority**(int newPriority)

Changes the priority of this thread. First the checkAccess method of this thread is called with no arguments. This may result in throwing a SecurityException. Otherwise, the priority of this thread is set to the smaller of the specified newPriority and the maximum permitted priority of the thread's thread group.

Parameters:

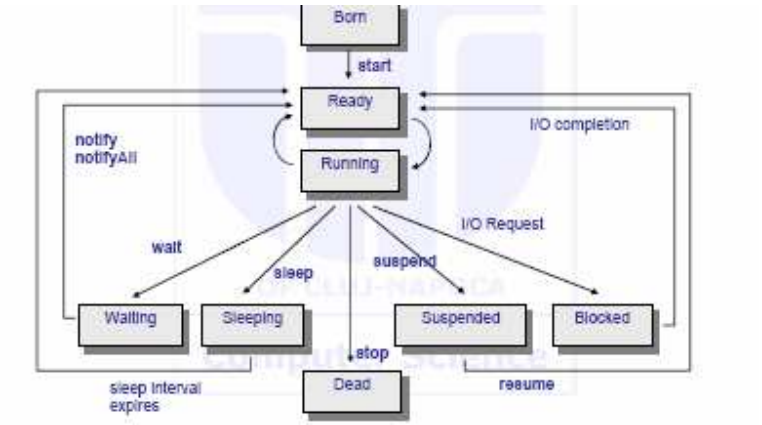
newPriority - priority to set this thread to

Throws:

IllegalArgumentException - If the priority is not in the range MIN_PRIORITY to MAX_PRIORITY.

SecurityException - if the current thread cannot modify this thread.

80.Describe the life cycle of a thread.



1. **New state -** After the creations of Thread instance the thread is in this state but before the start() method invocation. At this point, the thread is considered not alive.
2. **Runnable (Ready-to-run) state -** A thread start its life from Runnable state. A thread first enters runnable state after the invoking of start() method but a thread can return to this state after either running, waiting, sleeping or coming back from blocked state or suspended state. On this state a thread is waiting for a turn on processor.
3. **Running state -** A thread is in running state that means the thread is currently executing. There are several ways to enter in Runnable state but there is only one way to enter in Running state: the scheduler select a thread from runnable pool.
4. **Dead state -** A thread can be considered dead when its run() method completes. If any thread comes on this state that means it cannot ever run again.
5. **Blocked -** A thread can enter in this state because of waiting for a monitor lock.
6. **Waiting –** A thread that is waiting to be notified by another thread.

81.How should one correctly terminate a Java thread?

A thread terminates when its **run** method terminates

- Do not terminate a thread using the deprecated ***stop*** method
- Instead, notify a thread that it should terminate: **t.interrupt;**
- interrupt does not cause the thread to terminate – it sets a boolean field in the thread data structure

(The **run()** method should check occasionally whether it has been interrupted
An interrupted thread should release resources, clean up, and exit
The **sleep** method throws an **InterruptedException** when a sleeping thread is interrupted

82.What is a Java Collection?

Same as 52

83.What are the main restrictions which apply to applets?

Applets run inside a “sandbox”
Applet security manager protects the user
Applets can NOT:
- Access files from the client machine.
- Create files on the client machine.
- Make network connects except back to the host machine where the applet came from.
- Start any programs on a client machine.
- Load libraries.
- Define native method calls.
- Halt execution of the interpreter (can't call **System.exit()**).
- Can never run any client executable
- Cannot communicate with any host other than the server from which it came, “originating host”
- Cannot read or write to the client’s file system

Can find out only limited info about the client machine:
- Java version in use
- Name and version of OS running
- Characters used as file and line separators
- Client language (English) & locale (Eastern Europe)
- Client currency (Euro)

Windows popped up by an applet carry a warning message

84.What are the differences between a standalone application and an applet?

An instance of the applet is always created, and its constructor, init, and start methods are always run.

Because an Applet instance is a Panel instance (and JApplet inherits from Applet), a visible component is created, awt events are (potentially) handled, etc.

When a standalone application is invoked, only public static void main(String[]) (and code called by it) is run.

85.Describe the life cycle of an applet.

Every applet has 5 standard methods:
- **init()** — called by the browser when the applet is first loaded into memory
- **start()** — called by the browser to start animations running when the applet is made visible
- **stop()** — called by the browser to stop animations running when the applet is covered or minimized
- **destroy()** — called by the browser just before the applet is destroyed
- **paintComponent()** — called when the applet is drawn or re-drawn

Applet methods will always be called in the order **init**, **start**, **stop**, **destroy** – applet's life cycle **- start** and **stop** may be called many times during the life of an applet

All 5 methods are implemented as dummy methods in class **JApplet**, and the dummy methods are inherited by all applets. Applets override only the methods that they need to perform their function

86.What should be written in each of an applet's predefined methods?

The init() method performs the applet setup and contains initialization statements (this method is like a class constructor in a java application).

The start() method contains statements that do a 'revisit page' setup.

The stop() method contains clear-up or stop code.

(SURSA – NET:

The init() method is called exactly once in an applet's life, when the applet is first loaded. It's normally used to read PARAM tags, start downloading any other images or media files you need, and set up the user interface.

The start() method is called at least once in an applet's life, when the applet is started or restarted. It is often used to start any threads the applet will need while it runs.

The stop() method is called at least once in an applet's life, when the browser leaves the page in which the applet is embedded. Your applet should use the stop() method to pause any running threads.

The destroy() method is called exactly once in an applet's life, just before the browser unloads the applet. This method is generally used to perform any final clean-up.)

87.What are the top level containers for GUIs in Java?

JApplet is a top level Container.

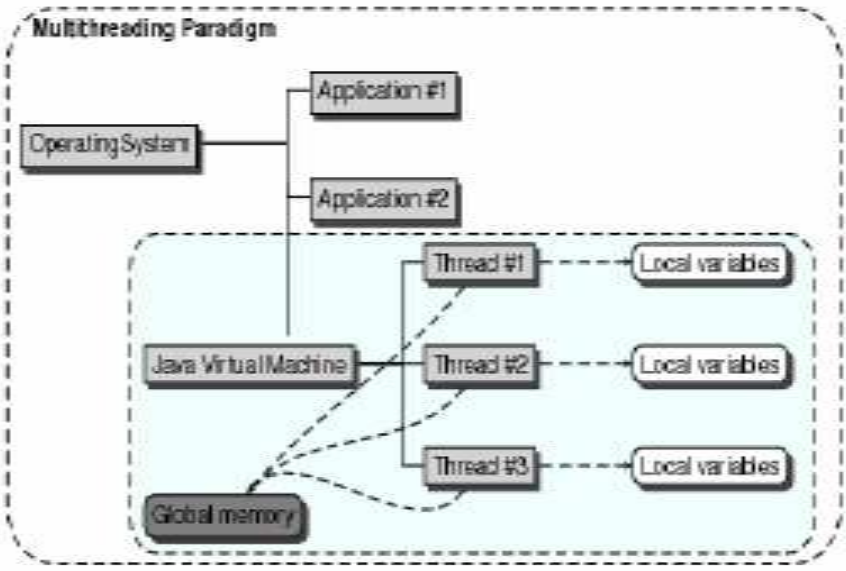
(A container is a graphical object that can hold components or other containers. The principal container is a Frame. It is a part of the computer screen surrounded by borders and title bars. Class Container has 2 principal descendent classes: JFrame and JPanel.)

88.What are Java threads?

Thread: a single sequential flow of control within a program (also called lightweight process). parallel processes running inside of a program

in Java you can create one or more threads within your program just as you can run one or more programs in an operating system

89.What are the differences between threads and operating system tasks?



The main difference consists in the fact that a single process can have multiple threads that share global data and address space with other threads running in the same process, and therefore can operate on the same data set easily. Processes do not share address space.

90.What is a factory method? Give a brief example.

A factory method is a static method which invokes a constructor. The constructor is usually private. The factory method can determine if to invoke or not the constructor. The factory method can throw an exception, or can do something else suitable, in case it is given illegal arguments or cannot create a valid object.

Ex:

```
public Person create(int age) {
    if (age>0) throw new IllegalArgumentException("Too young!");
    else return new Person(n);
    }
```