Oops in java

Overview and Characteristics of Java
Object-oriented programming organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data. An object-oriented program can be characterized as data controlling access to code

1. Abstraction refers to the act of representing essential features without including the background details or explanations.

2. Encapsulation is the mechanism that binds together code and the data it manipulates and keeps both safe from outside interference and misuse

A class defines the structure and behavior (data and code) that a set of objects will share. Each object of a given class contains the structure and behavior defined by the class
Objects are sometimes referred to as instances of a class
The data defined by the class are referred to as member variables or instance variables
The code that operates on that data is referred to as member methods or just methods

3. Inheritance is the process by which one object acquires the properties of another object.

4. Polymorphism (from Greek, meaning “many forms”) is a feature that allows one interface to be used for a general class of action

Compilation and Execution Process in Java
In Java, a source file is officially called a compilation unit. It is a text file that contains one or more class definitions. The Java compiler requires that a source file use the .java filename extension.
In Java, all code must reside inside a class. By convention, that class's name should match the file's name that holds the program. Java is case-sensitive.

C:\>javac Example.java

The javac compiler creates a file called Example.class that contains the bytecode version of the program. As discussed earlier, the Java bytecode is the intermediate representation of your program that contains instructions the Java Virtual Machine will execute. Thus, the output of javac is not code that can be directly executed. To run the program, you must use the Java application launcher, called Java.

C:\>java Example

When Java source code is compiled, each individual class is put into its own output file named after the class and using the .class extension.

public static void main(String args[])

- The public keyword is an access specifier, which allows the programmer to control the visibility of class members
- The keyword static allows main( ) to be called without having to instantiate a particular instance of the class.This is necessary since main( ) is called by the Java Virtual Machine before any objects are made.
- The keyword void simply tells the compiler that main( ) does not return a value
- String args[ ] declares a parameter named args, which is an array of instances of the class String. args receives any command-line arguments present when the program is executed.

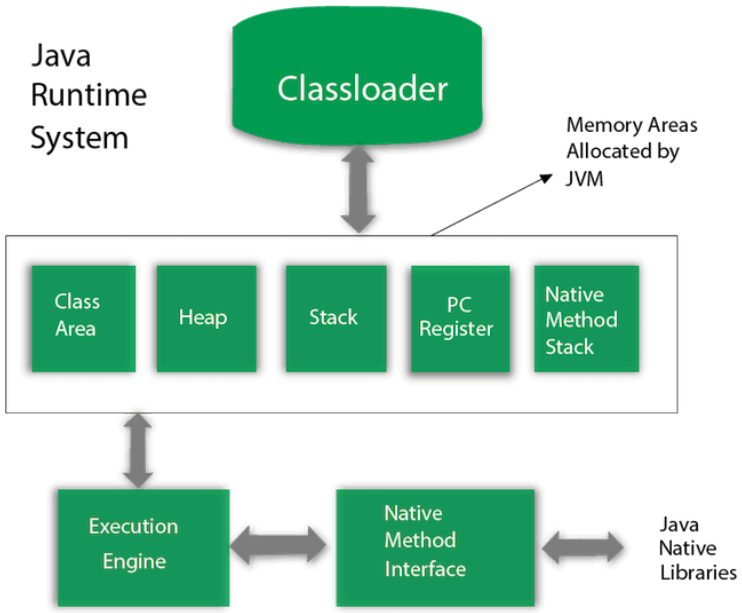
System.out.println("This is a simple Java program.");
System is a predefined class that provides access to the system, and out is the output stream that is connected to the console.. Output is actually accomplished by the built-in println( ) method

- All statements in Java end with a semicolon.
- Java is a free-form language. This means that you do not need to follow any special indentation rules.
- Identifiers are used for class names, method names, and variable names. An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters

the built-in method print( ) is used to display not followed by a newline. This means that when the next output is generated, it will start on the same line. The print( ) method is just like println( ), except that it does not output a newline character after each call.

Organization of the Java Virtual Machine

JVM(Java Virtual Machine) acts as a run-time engine to run Java applications
JVM is a part of JRE(Java Runtime Environment).



The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

Data types

- Integers: This group includes byte, short, int, and long, which are for whole-valued signed numbers.
- Floating-point numbers: This group includes float and double, which represent numbers with fractional precision
- Characters : This group includes char, which represents symbols in a character set, like letters and numbers.
- Boolean: This group includes boolean, which is a special type for representing true/false values.

String Class

Java implements strings as objects of type String
Strings are immutable , a new string object is created that contains the modification.
String are unchangeable means that the contents of the String instance cannot be changed after it has been created. However, a variable declared as a String reference can be changed to point at some other String object at any time.
The String, StringBuffer, and StringBuilder classes are defined in java.lang.
String s = new String();

char chars[] = { 'a', 'b', 'c' };
byte ascii[] = {65, 66, 67, 68, 69, 70 };
String s = new String(chars);
String s1 = new String(ascii);

char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
String s = new String(chars, 2, 3);
startIndex (2) specifies the index at which the subrange begins, and numChars(3) specifies the number of characters to use

char c[] = {'J', 'a', 'v', 'a'};
String s1 = new String(c);
String s2 = new String(s1);

String Operations

1. The length of a string is the number of characters that it contains.
   System.out.println(s.length());

2. String literals : *String s2 = "abc";*

3. + operator, which concatenates two strings,
   String age = "9";
   String s = "He is " + age + " years old.";
   System.out.println(s);

4. To implement toString( ), simply return a String object that contains the human-readable string that appropriately describes an object of your class.or example, they can be used in print( ) and println( ) statements and in concatenation expressions.
   // Override toString() for Box class.
   class Box {
   double width; double height; double depth;

```
Box(double w, double h, double d) { width = w; height = h; depth = d; }
public String toString()
{
        return "Dimensions are " + width + " by " + depth + " by " + height + "."; }
 }
 class toStringDemo
{
public static void main(String args[])
{       Box b = new Box(10, 12, 14);
        String s = "Box b: " + b; // concatenate Box object
        System.out.println(b); // convert Box to string
        System.out.println(s);
}
}
```

Box's toString( ) method is automatically invoked when a Box object is used in a concatenation expression or in a call to println( ).

5. **charAt( )** To extract a single character from a String, you can refer directly to an individual character via the charAt( ) method.

```
char ch;
ch = "abc".charAt(1);
```

6. To extract more than one character at a time, use the **getChars( )** method

```
public class Main
{
        public static void main(String[] args) {
           String S  = "This is a demo of the geChars method.";
           int start = 10;
           int end=14;
           char buf[]=new char[end-start];
           S.getChars(start,end,buf,0);
           System.out.println(buf);
        }
}
```

7. **getBytes( )** : stores the characters in an array of bytes

8. **toCharArray()** : to convert all the characters in a String object into a character array

```
public class Main
{
        public static void main(String[] args) {
           String S  = "This is a demo of the geChars method.";
           char buf[]=S.toCharArray();
           System.out.println(buf[0]);
```

```
        }
}
```

9. **equals( ) and equalsIgnoreCase( )**
   s1.equals(s2) It returns true if the strings contain the same characters in the same order, and false otherwise.
   The **== operator** compares two object references to see whether they refer to the same instance
   s1.equalsIgnoreCase(s4) it considers A-Z to be the same as a-z.
10. The **regionMatches( )** method compares a specific region inside a string with another specific region in another string. The index at which the comparison will start within str2 is specified by str2StartIndex. The length of the substring being compared is passed in numChars
    boolean regionMatches(int startIndex, String str2, int str2StartIndex, int numChars)
11. The **startsWith( )** method determines whether a given String begins with a specified string. Conversely, endsWith( ) determines whether the String in question ends with a specified string.
    "Foobar".endsWith("bar")
    "Foobar".startsWith("Foo")
    "Foobar".startsWith("bar", 3)
12. compareTo( ) less than, equal to, or greater than
    arr[i].compareTo(arr[j])
13. • indexOf( ) Searches for the first occurrence of a character or substring.
    • lastIndexOf( ) Searches for the last occurrence of a character or substring.
14. substring( ) : extract a substring using substring( )
    String substring(int startIndex, int endIndex)
15. concat( ) concatenate two strings
    String s1 = "one";
    String s2 = s1.concat("two");
16. replace() : replaces all occurrences of one character in the invoking string with another character. *String s = "Hello".replace('l', 'w');*
17. trim( ) method returns a copy of the invoking string from which any leading and trailing whitespace has been removed.
    String s = " Hello World ".trim();
18. The method toLowerCase( ) converts all the characters in a string from uppercase to lowercase. The toUpperCase( ) method converts all the characters in a string from lowercase to uppercase. Nonalphabetical characters, such as digits, are unaffected
19. boolean contains(CharSequence str)

String Buffer
StringBuffer represents growable and writeable character sequences. StringBuffer may have characters and substrings inserted in the middle or appended to the end

StringBuffer( )
StringBuffer(int size)
StringBuffer(String str)
StringBuffer(CharSequence chars)
The default constructor (the one with no parameters) reserves room for 16 characters without reallocation.

The current length of a StringBuffer can be found via the length( ) method, while the total allocated capacity can be found through the capacity( ) method.

ensureCapacity( ) preallocate room for a certain number of characters after a StringBuffer has been constructed, *void ensureCapacity(int capacity)*

setLength( ) To set the length of the buffer within a StringBuffer object, use setLength( ). I void *setLength(int len)*

charAt( ) and setCharAt( ) The value of a single character can be obtained from a StringBuffer via the charAt( ) method. You can set the value of a character within a StringBuffer using setCharAt( )

char charAt(int where)
void setCharAt(int where, char ch)

The append( ) method concatenates the string representation of any other type of data to the end of the invoking StringBuffer object. It has several overloaded versions.
StringBuffer append(String str)
StringBuffer append(int num)
StringBuffer append(Object obj)
E.g s = sb.append("a = ").append(a).append("!").toString(); the compiler inserts a call to toString( ) to turn the modifiable StringBuffer back into a constant String

The insert( ) method inserts one string into another. It is overloaded to accept values of all the simple types, plus Strings, Objects, and CharSequences. *sb.insert(2, "like ");*

To reverse the characters within a StringBuffer object using reverse( ), *s.reverse();*

The delete( ) method deletes a sequence of characters from the invoking object.The deleteCharAt( ) method deletes the character at the index specified by loc. It returns the resulting StringBuffer object
sb.delete(4, 7);
System.out.println("After delete: " + sb);
sb.deleteCharAt(0);

System.out.println("After deleteCharAt: " + sb);
replace one set of characters with another set inside a StringBuffer object by calling replace( ). substring( ) obtain a portion of a StringBuffer by calling substring( ). It has the following two forms: String substring(int startIndex) String substring(int startIndex, int endIndex)

Wrapper class : encapsulate a primitive type within an object.

A Wrapper class in Java is a class whose object wraps or contains primitive data types.
When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types.
They are passed by value to methods and cannot be directly passed by reference. Also, there is no way for two methods to refer to the same instance of an int

- There are 8 Wrapper classes in Java these are Boolean, Byte, Short, Character, Integer, Long, Float, Double.
- The automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing.
- It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing

```java
import java.util.ArrayList;
class Autoboxing {
    public static void main(String[] args)
    {
        char ch = 'a';

        // Autoboxing- primitive to Character object
        // conversion
        Character a = ch;

        ArrayList<Integer> arrayList
            = new ArrayList<Integer>();

        // Autoboxing because ArrayList stores only objects
        arrayList.add(25);

        // printing the values from object
        System.out.println(arrayList.get(0));
    }
}
```

```java
// Java program to demonstrate Unboxing
import java.util.ArrayList;

class Unboxing {
    public static void main(String[] args)
    {
        Character ch = 'a';

        // unboxing - Character object to primitive
        // conversion
        char a = ch;

        ArrayList<Integer> arrayList
            = new ArrayList<Integer>();
        arrayList.add(24);

        // unboxing because get method returns an Integer
        // object
        int num = arrayList.get(0);

        // printing the values from primitive data types
        System.out.println(num);
    }
}
```

Object Definitions:

- An object is a real-world entity.
- An object is a runtime entity.
- The object is an entity which has state and behavior.
- The object is an instance of a class.

A class is a group of objects which have common properties.

```
class Student
{
   int id;
   String name;
   Student()
   {
      id=00;
      name="not set";
   }
   void set(int i,String n)
   {
      id=i;
      name=n;
   }
   void get()
   {
      System.out.println("Student's id: " + id);
      System.out.println("Student's name "+ name);
   }
}
class Main
{
   public static void main(String args[])
   {
      Student S1 = new Student();
      Student S2 = new Student();
      Student S3 = new Student();
      S1.set(101,"Riya");
      S2.id=102;
      S2.name="Shyam";
      S1.get();
      S2.get();
      S3.get();
   }
}
```

this keyword.

this can be used inside any method to refer to the current object. That is, this is always a reference to the object on which the method was invoked.

finalize( ) method

The Java run time calls that method whenever it is about to recycle an object of that class. Inside the finalize( ) method, you will specify those actions that must be performed before an object is destroyed

final keyword

A variable can be declared as final. Doing so prevents its contents from being modified. This means that you must initialize a final variable when it is declared.o not occupy memory on a per-instance basis

extend keyword

To inherit a class, you simply incorporate the definition of one class into another by using the extends keyword

class subclass-name extends superclass-name {

```
        // body of class
}
```

Using super keyword

Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.

The first calls the superclass' constructor.

super(arg-list)

The second is used to access a member of the superclass that has been hidden by a member of a subclass.

```
class A {
        int i;
}
// Create a subclass by extending class A.
class B extends A
{       int i; // this i hides the i in A
        B(int a, int b) {
                super.i = a; // i in A
                i = b; // i in B }
        void show()
        {       System.out.println("i in superclass: " + super.i);
                System.out.println("i in subclass: " + i);
        }
 }
```

To call superclass methods

```
void show()
{
    //to call superclass show()
    super.show();
    System.out.println("Number3 is " + num2);
}
```

Multilevel hierarchy

abstract class

define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method.certain methods be overridden by subclasses by specifying the abstract type modifier. These methods are sometimes referred to as subclasser responsibility because they have no implementation specified in the superclass

Any class that contains one or more abstract methods must also be declared abstract

```
// A Simple demonstration of abstract.
abstract class A {
  abstract void callme();

  // concrete methods are still allowed in abstract classes
  void callmetoo() {
    System.out.println("This is a concrete method.");
  }
}

class B extends A {
  void callme() {
    System.out.println("B's implementation of callme.");
  }
}

class AbstractDemo {
  public static void main(String args[]) {
    B b = new B();

    b.callme();
    b.callmetoo();
  }
}
```

Final class

- **Using final to Prevent Overriding** To disallow a method from being overridden, specify final as a modifier at the start of its declaration. Methods declared as final cannot be overridden

```
class A {
  final void meth() {
    System.out.println("This is a final method.");
  }
}

class B extends A {
  void meth() { // ERROR! Can't override.
    System.out.println("Illegal!");
  }
}
```

- **Using final to Prevent Inheritance** to prevent a class from being inherited. To do this, precede the class declaration with final. Declaring a class as final implicitly declares all of its methods as final, too. As you might expect, it is illegal to declare a class as both abstract and final since an abstract class is incomplete by itself and relies upon its subclasses to provide complete implementations.

```
final class A {
  // ...
}

// The following class is illegal.
class B extends A { // ERROR! Can't subclass A
  // ...
}
```

Object class

There is one special class, Object, defined by Java. All other classes are subclasses of Object. That is, Object is a superclass of all other classes. This means that a reference variable of type Object can refer to an object of any other class

Java Object Class - Javatpoint

Packages and interfaces

Packages are containers for classes that are used to keep the class name space compartmentalized.The package statement defines a name space in which classes are stored. The package is both a naming and a visibility control mechanism. You can define classes inside a package that are not accessible by code outside that package. You can also define class members that are only exposed to other members of the same package

Java Package - javatpoint

Using the keyword interface, you can fully abstract a class' interface from its implementation. That is, using interface, you can specify what a class must do, but not how it does it. Interfaces are syntactically similar to classes, but they lack instance variables, and their methods are declared without any body.

By providing the interface keyword, Java allows you to fully utilize the “one interface, multiple methods” aspect of polymorphism
[Interface in Java - Javatpoint]

Access protection / control
Public : Classes, methods, or data members that are declared as public are accessible from everywhere in the program. There is no restriction on the scope of public data members.
Private : The methods or data members declared as private are accessible only within the class in which they are declared
Protected : The methods or data members declared as protected are accessible within the same package or subclasses in different packages

	Default	Private	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclass	Yes	No	Yes	Yes
Same Package Non-Subclass	Yes	No	Yes	Yes
Different Package Subclass	No	No	Yes	Yes
Different Package N0on-Subclass	No	No	No	Yes

Multithreading

A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.

Process-based multitasking is the feature that allows your computer to run two or more programs concurrently. For example, process-based multitasking enables you to run the Java compiler at the same time that you are using a text editor. In process based multitasking, a program is the smallest unit of code that can be dispatched by the scheduler.

Thread-based multitasking environment, the thread is the smallest unit of dispatchable code. This means that a single program can perform two or more tasks simultaneously. For instance, a text editor can format text at the same time that it is printing, as long as these two actions are being performed by two separate threads

In a Multithreaded program with a single processor, the processor divides the execution time equally amongst all the running threads.Thus each thread gets the processor attention in a round robin manner. Once the time-slice allocated for a thread expires, the operation that is currently being performed is put on hold and the processor now directs its attention to the next thread. Thus, at any given moment, if we take the snapshot of memory, only one thread is being executed by the processor. The switching of attention from one thread to another happens so fast that we get the effect as if the processor is exciting several threads simultaneously.

Advantages: Responsiveness, Simplifies program organization, improving the performance

Thread is a predefined class in java. It is a basic unit of CPU and it is well known for independent execution.
To create a new thread, your program will either extend Thread or implement the Runnable interface

Method	Meaning
getName	Obtain a thread's name.
getPriority	Obtain a thread's priority.
isAlive	Determine if a thread is still running.
join	Wait for a thread to terminate.
run	Entry point for the thread.
sleep	Suspend a thread for a period of time.
start	Start a thread by calling its run method.

```
class Main
{
   public static void main(String args[])
   {
      Thread t=Thread.currentThread();
      System.out.println("Current thread: " + t);
      t.setName("My thread");
      System.out.println("Current thread: " + t);
      System.out.println("thread name: " + t.getName());
      System.out.println("thread priority: " + t.getPriority());
      System.out.println("thread is alive: " + t.isAlive());
   }
}
```

OUTPUT
Current thread: Thread[main,5,main]
Current thread: Thread[My thread,5,main]
thread name: My thread
thread priority: 5
thread is alive: true

the name of the thread, its priority, and the name of its group

Java thread model:
Thread classes and Runnable interface
Creating a thread in Java - javatpoint
Multi threading
Thread priorities: Thread priorities are used by the thread scheduler to decide when each thread should be allowed to run. In theory, higher-priority threads get more CPU time than lower-priority threads.
To set a thread's priority, use the setPriority( ) method, which is a member of Thread
final void setPriority(int level)
The value of level must be within the range MIN_PRIORITY and MAX_PRIORITY. Currently, these values are 1 and 10, respectively. To return a thread to default priority, specify NORM_PRIORITY, which is currently 5. These priorities are defined as static final variables within Thread.

Thread Synchronization:
When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called synchronization.
Key to synchronization is the concept of the monitor (also called a semaphore). A monitor is an object that is used as a mutually exclusive lock, or mutex. Only one thread can own a monitor at a given time. When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor. These other threads are said to be waiting for the monitor. A thread that owns a monitor can reenter the same monitor if it so desires

Using Synchronized Methods
To enter an object's monitor, just call a method that has been modified with the synchronized keyword. While a thread is inside a synchronized method, all other threads that try to call it (or any other synchronized method) on the same instance have to wait. To exit the monitor and relinquish control of the object to the next waiting thread, the owner of the monitor simply returns from the synchronized method.

The synchronized block
At times it may so happen that a class has been developed with a view,to use it in a single thread situation. But later on a need arises to use it in a multithreaded situation. If we do not have access to its code, we cannot mark the methods in it as synchronized. In such situations, a synchronized block is used.

Inter thread communication
Polling is usually implemented by a loop that is used to check some condition repeatedly. Once the condition is true, appropriate action is taken. This wastes CPU time. For example, consider the classic queuing problem, where one thread is producing some data and another is

consuming it. To make the problem more interesting, suppose that the producer has to wait until the consumer is finished before it generates more data

Java includes an elegant interprocess communication mechanism that allow threads to communicate with each other via the **wait( ), notify( ), and notifyAll( )** methods. These methods are implemented as final methods in Object, so all classes have them. All three methods can be called only from within a synchronized context

- wait( ) tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).
- notify( ) wakes up a thread that called wait( ) on the same object.
- notifyAll( ) wakes up all the threads that called wait( ) on the same object. One of the threads will be granted access.

These methods are declared within Object, as shown here:
final void wait( ) throws InterruptedException
final void notify( )
final void notifyAll()

Suspending, Resuming and stopping threads
Thread lifecycle
Life cycle of a thread in Java - javatpoint