JAVA NOTES MADE HOURS BEFORE EXAM

- Separators: parentheses, ( ), braces, { }, the period, ., and the semicolon, ;
- java compiler converts the source code(.java) into byte code(.class)
- java is case sensitive
- Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other
- Cohesion refers to the level of a component which performs a single well-defined task. A single welldefined task is done by a highly cohesive method
- Association represents the relationship between the objects
- Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state.
- The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state.
- JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode
- JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.
- JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.
- Object - An entity that has state and behavior. Its an instance of class
- Instance variable
    - A variable which is created inside the class but outside the method
    - It doesn't get memory at compile time. It gets memory at runtime when an object or instance is created.
- Method - function of java
- new - allocate memory at runtime
- Anonymous object aka nameless object
    - Calculation c=new Calculation(); c.fact(5); //normal
    - new Calculation().fact(5); //anonymous
- BASIC PROGRAM
    - class Simple{

        public static void main(String args[]){

```
                System.out.println("Hello Java");
            }
        }
```

- Operators: a symbol which is used to perform operations +, -, *, /
- Operator types:
    - Unary Operator(a++,a--)
    - Arithmetic Operator(a+b,a*b,a-b,a/b)
    - Shift Operator
        - Left shift (10<<5) means 10*2^5
        - Right shift (20>>3) means 20/2^3
    - Relational Operator(<,>,==)
    - Bitwise Operator(&,|) always checks both
    - Logical Operator(&& will not check second if first is false, || will not check second if first is true)
    - Ternary Operator (a<b)?a:b
    - Assignment Operator(=)
- type cast a=(short)(a+b); a will be short now
- constructor - method which is called when an instance of the class is created
    - Type 1 - Default constructor
    - Type 2 - Parameterised constructor
    - name must be same as the class name
    - constructor overloading - different constructor with same name but different parameter list
- literals: identifiers(integer,decimal,floating-point,character identifier etc)
- Escape sequences: signal an alternative interpretation of a series of characters eg if \ is before then it becomes escape sequence like \n for new line
- Initializing a variable at run time is called dynamic initialization
- Control statements
    - if(condition) { //shit }
    - if(condition) { //shit } else { //alternate shit }
    - if-else-if
    - try{ //naughty code which can throw exception } catch(//naughtyness aka exception){ //do something with it }
    - for(i=0;i<9;i++){ //shit }
    - for-each eg. for(Type var:array)
        - int i=0; int a[]={3,5,7,4};
        - for(i:a){ //shit }
    - labeled for eg.
        - num: //label
        - for(i=0;i<5;i++){

```
//shit
num2:
for(j=0;j<9;j++){ break num; }
```

    - }
  - switch(expression) { case value1: //shit; break; case value2: //shit; default: //shit}
  - while(a<b){ //Ah, shit }
  - do{ //shit }while(a<b)
- break statement terminates the loop when encountered
- continue statement jumps to the next iteration of the loop
- return completes execution of method
- this - to refer current class instance variable
- Array
  - collection of similar data types
  - eg. datatype[] arrayName; or datatype arrayName[]; //single dimention
  - eg. datatype[][] arrayName; or datatype arrayName[][]; //double dimention
- Access Modifiers
  - specifies the visibility or scope of variables and methods of a class
  - default access modifier: they can be accessed by any other classes in the same package
  - public access modifier: can be accessed from any other class
  - private access modifier: private variables or methods cannot be accessed even by the objects of the same class. They can be accessed only by the public methods of that class.
  - protected access modifier: accessible to all classes and subclasses in the same package and also to subclasses in the other packages.
- Inheritance
  - mechanism that allowed a class to inherit property of another class.
  - inherits all non-private members including fields and methods.
  - extends keywords are used to describe inheritance
  - Eg of extends
    - class Vehicle { ..... } //super class
    - class Car extends Vehicle { ..... } //sub class
  - Types
    - Single
    - Multilevel
    - Hierarchical
  - Note: multiple inheritance not supported in java
  - super is used to refer the parent
  - Makes super and sub tightly coupled

- Upcasting
    - If the reference variable of Parent class refers to the object of Child class
    - Eg.
        - class A { }
        - class B extends A{ }
        - A a=new B(); //upcasting
- Polymorphism
    - single action in different ways
    - Types
        - Compile time - overload a static method
        - Runtime or Dynamic Method Dispatch - an overridden method is called through the reference variable of a superclass aka by using upcasting
- Lambda expression
    - It is a function that has no name and uses a functional approach to execute code
    - also known as an anonymous function aka nameless
    - Eg.
        - (list of arguments) -> { expression body}
        - (int a, int b) -> { a+b; }
- Final Modifier
    - declare a field as final aka constant
    - it prevents its content from being modified
    - final int a = 10;
    - Final Class
        - Class declared with final
        - To prevent inheritance
    - Final Method
        - method which is declared using final keyword
        - when we want to prevent a method from overridden.
- Static
    - used for memory management
    - The static variable gets memory only once in the class
    - A static method belongs to the class rather than the object of a class.
    - A static method can be invoked without the need for creating an instance of class.
    - A static method can access static data member and can change the value of it.
    - Restrictions on static method
        - this and super cannot be used in static context.

- Static Blank Final Variable
    - used to create static constant for the class.
    - can be initialized in static block only
- Method overriding vs method overloading

Declaring a method in sub class which is already present in parent class	Same name but different parameter list

- Abstract class
    - class that is declared using “abstract” keyword
    - It can have abstract methods (methods without body) as well as concrete methods (regular methods with body).
- Java garbage collection
    - process of releasing unused memory occupied by unused objects
    - done by the JVM automatically
    - When a Java programs run on the JVM, objects are created on the heap. When no longer needed they are discarded
- Interface
    - used to achieve abstraction
    - similar to classes but cannot make objects
    - can have only abstract methods and static fields
    - It summons java to support multiple inheritance
    - can be used to achieve loose coupling.
    - Eg.
        - Interface inter {
            public String disptext(); /no implementation of method
            }
            public class imp implements inter {
            public String disptext() {
            return “I love java”; //declaration
            }
            }
    - Multiple inheritance is done by a class by implementing multiple interfaces
- Package
    - collection of related class, interfaces, sub-packages
    - are used to avoid name conflicts
    - Is used to categorize the classes and interface so that they can be easily maintained.
    - provides access protection
    - Can be built in like util, lang, awt
    - Can be user defined eg.
        - package mypack;

        - public class employee {
            String empId;
            String name;
          }
    - Can be imported in java program by
        - Importing specific class eg. import packg.abc;
        - Import all classes eg. import java.awt.*;
        - Without import if we use fully qualified names
            - Eg.
            - package pack; //saved in A.java
              public class A {
                  public void msg() {
                      System.out.println("Hello");
                  }
              }
              package mypack; ///saved in B.java
              class B {
                  public static void main(String args[]) {
                      pack.A obj = new pack.A();
                      obj.msg();
                  }
              }
- Program in execution = process
- Thread aka lightweight sub-process
    - Multiple can run simultaneously
    - Process of running multiple threads = multithreading
    - Share common memory space
    - main thread is created automatically when your program is started
    - actual code which represents the functionality of a thread is introduced by a run() method and it is also the entry point for that thread.
    - Creating a thread
        - Implementing runnable interface
            - class abc implements Runnable {
                  public void run(){
                      System.out.println("Thread started its execution");
                      for(int i=1;i<=10;i++) System.out.println(i+"* 5 ="+ i*5);
                      System.out.println("Thread completed its execution");

```
        }
    }
    public class firstthread{
        public static void main(String args[]) {
            abc r = new abc();
            Thread t= new Thread(r);
            t.start();
        }
    }
```

- Extending thread class
  - ```
    class MyThread extends Thread {
        public void run() {
            System.out.println("concurrent thread started
            running..");
            }
        }
    classMyThreadDemo {
        public static void main(String args[]) {
            MyThread mt = new MyThread();
            mt.start();
            mt.sleep(4000);
        }
    }
    ```

- Some thread methods
  - setName() to give thread a name
  - getName() return thread's name
  - getPriority() return thread's priority
  - isAlive() checks if thread is still running or not
  - join() Wait for a thread to end
  - run() Entry point for a thread
  - sleep() suspend thread for a specified time
  - start() start a thread by calling run() method
  - currentThread() Returns a reference to the currently executing thread object.
  - getId() Returns the identifier of this Thread.
  - getState() Returns the state of this thread.
  - interrupt() Interrupts this thread.
  - isInterrupted() Tests whether this thread has been interrupted.
  - Suspend() Suspend the thread
  - Resume() resume the thread
    ```

- Thread Priority
  - thread scheduler schedules the threads according to priority
  - NORM_PRIORITY (default) (5)
  - MIN_PRIORITY (1)
  - MAX_PRIORITY (10)
  - Eg. m2.setPriority(Thread.MAX_PRIORITY);
- Synchronization
  - capability to control the access of multiple threads to any shared resource.
  - why?
    - To remove consistency problem
    - To remove thread interference
  - synchronized (object) { //statement to be synchronized }
- Thread control
  - suspend() to suspend
  - stop() to stop a thread
  - resume() to resume which was suspended
  - wait() causes the current thread to wait until another thread invokes the notify().
  - notify() Wakes up a single thread that is waiting on this object's monitor.

- Files
  - mechanism for storing data
  - They typically give IOExceptions
  - defined in java.io
  - constructor for file object
    - File(string dirpath) //dirpath is pathname of the file
    - File(String dirpath, String filename) //Filename is name of the file
    - File (File dirobj, String Filename) //dirobj is file object that specifies a directory
  - Methods of file class
    - getName() Returns name of file/directory
    - getParent() Returns the pathname string of the parent
    - getParentFile() Returns the file of the parent
    - getPath() Converts abstract pathname into a pathname string.
    - isAbsolute() Tests whether this abstract pathname is absolute.
    - getAbsolutePath() Returns the absolute pathname string
  - Writing to a file
    - FileWriter f = new FileWriter(“test.txt”);
    - PrintWriter out=new PrintWriter(f);

        - Out.println(“some text write to the file”);
        - out.close();
        - f.close();
    - Reading from a file
        - FileReader f = new Filereader(“test.txt”);
        - BufferedReader in =new BufferedReader(f);
        - String line =in.readLine();
        - f.close();
    - Deleting a file
        - File f1 = new File (“shc.txt”);
        - Boolean b= f1.delete();
        - //if b is true, then the file has been deleted successfully
    - Renaming a file
        - File f1 = new File(“oldname.txt”);
        - File f2 = new File (“newname.txt”);
        - Boolean b = f1.renameTo(f2);
        - If b is true, then the file has been renamed successfully.
        - If newname.txt file exist then, it will be overwritten
- Streams
    - A stream can be defined as a sequence of data
    - InPutStream − The InputStream is used to read data from a source.
    - OutPutStream − The OutputStream is used for writing data to a destination.
    - Byte Stream - Java byte streams are used to perform input and output of 8-bit bytes
        - FileInputStream and FileOutputStream.
        - Eg.

```
FileInputStream in = null;
FileOutputStream out = null;
try {
        in = new FileInputStream("input.txt");
        out = new FileOutputStream("output.txt");
        int c;
        while ((c = in.read()) != -1) {
                out.write(c);
        }
}finally {
        if (in != null) { in.close(); }
        if (out != null) { out.close(); }
}
```

        - FileInputStream
            - File f = new File("C:/java/hello");
            - InputStream f = new FileInputStream(f);
            - Methods
                - close() This method closes the file output stream.
                - finalize() This method cleans up the connection to the file.
                - read(int r) reads the specified byte of data from the InputStream. Returns an int. Returns the next byte of data and -1 will be returned if it's the end of the file.
                - read(byte[ ] r) reads r.length bytes from the input stream into an array. Returns the total number of bytes read.
                - available() Gives the number of bytes that can be read from this file input stream. Returns an int.
        - FileOutputStream
            - File f = new File("C:/java/hello");
            - InputStream f = new FileOutputStream(f);
            - Methods
                - close() This method closes the file output stream.
                - finalize() This method cleans up the connection to the file
                - write(int w) writes the specified byte
                - write(byte[ ] w) Writes w.length bytes from the mentioned byte array to the OutputStream.

    - Character Streams - Java Character streams are used to perform input and output for 16-bit unicode.
        - FileWriter
        - FileReader
        - Eg.

```
FileReader in = null;
FileWriter out = null;
try {
        in = new FileReader("input.txt");
        out = new FileWriter("output.txt");
        int c;
        while ((c = in.read()) != -1) {
                out.write(c);
        }
```

```
        }finally {
            if (in != null) { in.close(); }
            if (out != null) { out.close(); }
        }
```

  - Standard Streams
    - System.in
    - System.out
    - System.err
- Inputing Data we can use
  - Scanner
  - BufferedReader
  - DataInputStream
- Scanner
  - found in the java.util package
  - read input from the keyboard
  - breaks the input into tokens using a delimiter which is whitespace by default
  - Scanner a= new Scanner (System.in).useDelimiter("\\s"); \s is whitespace
  - provides nextXXX() methods e.g. nextInt(), nextDouble(), nextLine etc.
  - Theres no next char although, workaround is a.next().charAt(0);
- DataInputStream
  - in java.io
  - Uses readLine( ) for input
  - Gets a string so we need to parse data if needed
  - Eg
    - DataInputStream dis = new DataInputStream(System.in);
    - String str1 = dis.readLine();
- BufferedReader
  - In java.io
  - buffers the characters so that it can get the efficient reading
  - Gets a string so we need to parse data if needed
  - Eg
    - InputStreamReader r= new InputStreamReader(System.in);
    - BufferedReader o = new BufferedReader(r);
- BufferedReader v/s Scanner

larger buffer memory(8KB or 8192 chars)	Low (1KB or 1024 chars)
is used to read the data only	read the data but also parse the data.
faster	slower

synchronized	Not synchronized
in java.io	in java.util

- AWT
    - Require java.awt.*;
    - Frame f=new Frame("Button Example");
        - Button => Button b=new Button("Click Here");
        - Label => Label l1; l1=new Label("First Label.");
        - TextField => TextField t1; t1=new TextField("Welcome");
        - TextArea => TextArea area=new TextArea("Welcome");
        - Checkbox => Checkbox checkbox1 = new Checkbox("C++",true);
        - CheckboxGroup => CheckboxGroup cbg = new CheckboxGroup(); Checkbox checkBox1 = new Checkbox("C++", cbg, false);
        - Choice => Choice c=new Choice(); c.add("Item 1"); c.add("Item 2");
        - List => List l1=new List(5); l1.add("Item 1"); l1.add("Item 2");
        - Scrollbar => Scrollbar s=new Scrollbar();
        - MenuItem =>
            - MenuBar mb=new MenuBar();
            - Menu menu=new Menu("Menu");
            - Menu submenu=new Menu("Sub Menu");
            - MenuItem i1=new MenuItem("Item 1");
            - MenuItem i2=new MenuItem("Item 2");
            - MenuItem i3=new MenuItem("Item 3");
            - MenuItem i4=new MenuItem("Item 4");
            - MenuItem i5=new MenuItem("Item 5");
            - menu.add(i1);
            - menu.add(i2);
            - menu.add(i3);
            - submenu.add(i4);
            - submenu.add(i5);
            - menu.add(submenu);
            - mb.add(menu);
            - f.setMenuBar(mb);
        - Panel =>
            - simplest container class. It provides space in which an application can attach any other component
            - Panel panel=new Panel();
            - panel.setBounds(40,80,200,200);
            - panel.setBackground(Color.gray);
            - panel.add(b1); panel.add(b2);

                - f.add(panel);
            - Dialog => d = new Dialog(f , "Dialog Example", true);
            - Canvas =>

```
f.add(new MyCanvas());
class MyCanvas extends Canvas {
    public MyCanvas() {
        setBackground (Color.GRAY);
        setSize(300, 200);
    }
    public void paint(Graphics g) {
        g.setColor(Color.red);
        g.fillOval(75, 75, 150, 75);
    }
}
```

    - b.setBounds(50,100,80,30);
    - f.add(b); f.setSize(400,400); f.setLayout(null); f.setVisible(true);
- Layout manager
    - used to arrange components in a particular manner
    - BorderLayout
        - arrange the components in five regions: north, south, east, west and center
        - default layout of frame or window
        - Constants
            - NORTH
            - SOUTH
            - EAST
            - WEST
            - CENTER
        - Constructors
            - BorderLayout(): no gaps
            - JBorderLayout(int hgap, int vgap)
    - GridLayout
        - Arrange in rectangular grid
        - Constructors
            - GridLayout()
            - GridLayout(int rows, int columns)
            - GridLayout(int rows, int columns, int hgap, int vgap)
    - FlowLayout
        - arrange the components in a line, one after another
        - default layout of applet or panel.

        - Constructors
            - FlowLayout() 5 unit horizontal and vertical gap
            - FlowLayout(int align) 5 unit horizontal and vertical gap.
            - FlowLayout(int align, int hgap, int vgap)
        - Fields
            - LEFT
            - RIGHT
            - CENTER
            - LEADING
            - TRAILING
    - BoxLayout
        - arrange the components either vertically or horizontally
        - Constructor
            - BoxLayout(Container c, int axis)
        - Constants
            - X_AXIS
            - Y_AXIS
            - LINE_AXIS
            - PAGE_AXIS
    - CardLayout
        - manages the components in such a manner that only one component is visible at a time.
        - Constructor
            - CardLayout()
            - CardLayout(int hgap, int vgap):
        - methods
            - next(Container parent): is used to flip to the next card of the given container.
            - previous(Container parent): is used to flip to the previous card of the given container.
            - first(Container parent): is used to flip to the first card of the given container.
            - last(Container parent): is used to flip to the last card of the given container.
            - show(Container parent, String name): flip to the specified card with the given name.
    - GridBagLayout
        - align components vertically, horizontally or along their baseline
    - GroupLayout groups its components and places them in a Container hierarchically

    - A SpringLayout arranges the children of its associated container according to a set of constraints
- Event Handling
    - mechanism that controls the event and decides what should happen if an event
    - Event
        - Change in the state of an object
        - example, clicking on a button,
        - Types
            - Foreground - require user
            - Background - dont require user
    - Java Uses the Delegation Event Model
        - Source - object on which event occurs
        - Listener - responsible for generating response
        - **Listener needs to be registered with the source
        - Callback Methods - methods that are provided by API. provided in listener interfaces
    - Requires java.awt.event package which has
        - Event class - listener interface
            - ActionEvent - ActionListener
            - MouseEvent - MouseListener and MouseMotionListener
                - MouseListener methods
                    - mouseClicked()
                    - mouseEntered()
                    - mouseExited()
                    - mousePressed()
                    - mouseReleased()
                - MouseMotionListener methods
                    - mouseDragged()
                    - mouseMoved()
            - MouseWheelEvent - MouseWheelListener
            - KeyEvent - KeyListener
                - Methods
                    - keyPressed
                    - keyReleased
                    - keyTyped
            - ItemEvent - ItemListener
                - Method => itemStateChanged
            - TextEvent - TextListener
            - AdjustmentEvent - AdjustmentListener

- WindowEvent - WindowListener
- ComponentEvent - ComponentListener
- ContainerEvent - ContainerListener
- FocusEvent - FocusListener

- Registration methods
  - addActionListener(ActionListener a) {}
    - Button
    - MenuItem
    - TextField
    - List
      - Implement the ActionListener interface
      - Eg. public class ActionListenerExample Implements ActionListener
      - Register the component
      - Eg. component.addActionListener(this)
      - Override the actionPerformed()
      - Eg. public void actionPerformed(ActionEvent e){ }
  - addTextListener(TextListener a) {}
    - TextField
    - TextArea
  - addItemListener(ItemListener a) {}
    - Checkbox
    - Choice
    - List

- Examples
  - By implementing ActionListener
    - ```
      import java.awt.*;
      import java.awt.event.*;
      class AEvent extends Frame implements ActionListener{
          TextField tf;
          AEvent(){
              tf=new TextField();
              tf.setBounds(60,50,170,20);
              Button b=new Button("click me");
              b.setBounds(100,120,80,30);
              b.addActionListener(this);
              add(b);add(tf);
              setSize(300,300);
              setLayout(null); setVisible(true);
      ```
      ```

```
        }
        public void actionPerformed(ActionEvent e){
            tf.setText("Welcome");
        }
        public static void main(String args[]){ new AEvent(); }
    }
```

- By outer class

```
    import java.awt.*;
    import java.awt.event.*;
    class AEvent2 extends Frame{
        TextField tf;
        AEvent2(){
            tf=new TextField();
            tf.setBounds(60,50,170,20);
            Button b=new Button("click me");
            b.setBounds(100,120,80,30);
            Outer o=new Outer(this);
            b.addActionListener(o);
            add(b);add(tf);
            setSize(300,300);
        setLayout(null);
        setVisible(true);
        }
        public static void main(String args[]){ new AEvent2(); }
    }
    class Outer implements ActionListener{
        AEvent2 obj;
        Outer(AEvent2 obj){ this.obj=obj; }
        public void actionPerformed(ActionEvent e){
            obj.tf.setText("welcome");
        }
    }
```

- by anonymous class

```
    import java.awt.*;
    import java.awt.event.*;
    public class ActionListenerExample {
        public static void main(String[] args) {
        Frame f=new Frame("ActionListener Example");
        final TextField tf=new TextField();
        tf.setBounds(50,50, 150,20);
```

```
                Button b=new Button("Click Here");
                b.setBounds(50,100,60,30);
                b.addActionListener(new ActionListener(){
                        public void actionPerformed(ActionEvent e){
                                tf.setText("Welcome to Javatpoint.");
                        }
                });
                f.add(b);
                f.add(tf);
                f.setSize(400,400);
                f.setLayout(null);
                f.setVisible(true);
                }
        }
```

- Applet
    - small windows-based program
    - designed to be executed within an HTML web document using an external API.
    - Java-enabled web browser needed
    - Compiled using javac
    - Can be executed with appletviewer or browser
    - needs the AWT for interaction with the user by way of some GUI components
    - Types
        - Awt based
        - Swing based
    - Applet class provides all necessary support for applet execution, such as initializing and destroying of applet, methods etc.
        - init() : This is where variable are initialized
        - start() : called after init(). is called to restart an applet after it has been stopped
        - stop() : stop() method is called to suspend a thread that does not need to run when the applet is not visible.
        - destroy() : destroy() method is called when your applet needs to be removed completely from memory
        - Note: stop() method is always called before destroy() method.
    - Graphics
        - java.awt.Graphics is needed
        - Methods
            - drawString(String str, int x, int y): draws string.

- drawRect(int x, int y, int width, int height): draws a rectangle
- void fillRect(int x, int y, int width, int height): is used to fill rectangle with the default color and specified width and height.
- drawOval(int x, int y, int width, int height): draw oval
- void fillOval(int x, int y, int width, int height): fill oval with the default color
- drawLine(int x1, int y1, int x2, int y2): is used to draw line between the points(x1, y1) and (x2, y2).
- drawImage(Image img, int x, int y, ImageObserver observer): draw the specified image.
- drawArc(int x, int y, int width, int height, int startAngle, int arcAngle): draw a circular or elliptical arc.
- void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): fill a circular or elliptical arc.
- setColor(Color c): set the graphics current
- setFont(Fontfont): set the graphics current font to the specified font.

  - Eg.

```
import java.applet.*;
import java.awt.*;
public class demo extends applet {
        public void paint(Graphics g) {
        g.drawString("welcome",100,100);
        }
 }
```

- Applications
  - stand-alone Java program that runs with the support of a virtual machine in a client or server side
- Applications v/s Applet

Run standalone	Run in html
Require main method()	Require html file with java enabled browser
can access all the resources of the system including data and information	cannot access or modify any resources on the system except only the browser specific
do not require any security as they are trusted	cannot be run independently, thus require highest level of security

- Maven
    - Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation.
- Annotations
    - Annotation is a tag that represents the metadata i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.
    - Eg. of built-in
        - @Override - assures that the subclass method is overriding the parent class method
        - @SuppressWarnings - used to suppress warnings issued by the compiler.
        - @Deprecated - marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.
    - Custom Annotations
        - @interface MyAnnotation{}
    - Built-in Annotations used in custom annotations
        - @Target - specify at which type, the annotation is used
            - Types can be
                - TYPE class, interface or enumeration
                - FIELD
                - METHOD
                - CONSTRUCTOR
                - LOCAL_VARIABLE
                - ANNOTATION_TYPE
                - PARAMETER
            - eg.
              @Target({ElementType.TYPE,
              ElementType.FIELD,ElementType.METHOD})
              @interface MyAnnotation{
              int value1();
              String value2();
              }
        - @Retention - specify to what level annotation will be available.
            - Eg.
            - @Retention(RetentionPolicy.RUNTIME) //or SOURCE or CLASS
        - @Inherited - marks the annotation to be inherited to subclasses.

            - Default no inheritance
            - @Inherited
            - @interface ForEveryone { }//Now it will be available to subclass also
        - @Documented - Marks the annotation for inclusion in the documentation.
- Strings
    - string is basically an object that represents sequence of char values
    - java.lang.String is required
    - Methods
        - charAt(int index) - returns char value for the particular index
        - length() - returns string length
        - concat() - for concatenation
        - substring(int startIndex) - get the substring from start index
        - substring(int startIndex, endindex) - get the substring from start index to endindex
        - toUpperCase() - converts this string into uppercase letter
        - toLowerCase() - converts into lowercase letter.
        - trim() - method eliminates white spaces before and after string
        - startsWith() - checks if a string starts with
        - endsWith() - checks if a string ends with
        - valueOf() - coverts given types such as int, long, float, double, boolean, char and char array into string.
        - replace(“java”,”kotlin”) - replaces all occurrence of first sequence of character with second sequence of character
        - equals() - compares the original content of the string
        - compareTo() - compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string
    - mutable string - A string that can be modified or changed
        - StringBuilder
            - Not synchronized
            -
        - StringBuffer
            - Its synchronized and thread safe
            - StringBuffer sb=new StringBuffer("Hello ");
            - Methods
                - sb.append("Java");
                - sb.insert(1,"Java");
                - sb.replace(1,3,"Java");
                - sb.delete(1,3);
                - sb.reverse();
                - sb.capacity(); //current capacity of buffer
                -

    - StringTokenizer
        - is used to break a string into tokens based on provided delimiter
        - StringTokenizer(String str)
        - StringTokenizer(String str, String delim)
        - StringTokenizer(String str, String delim, booleanreturnValue)
- Exception
    - mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote
    - the advantage of exception handling is to maintain the normal flow of the application.
    - Types
        - Checked - classes that extend Throwable class except RuntimeException and Error are known as checked exceptions. Checked exceptions are checked at compile-time.
        - Unchecked - The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.
        - Error - Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.
    - Keywords
        - try
        - catch
        - finally - finally block is a block that is used to execute important code such as closing connection, stream etc. Java finally block is always executed whether exception is handled or not. Java finally block must be followed by try or catch block.
        - throw - used to explicitly throw an exception
        - throws - is used to declare an exception
    - User-defined exception
        - class UserDefinedException extends Exception { // code }
        - class MarksOutOfBoundsException extends Exception
        - {
        - public MarksOutOfBoundsException(String str)
        - {
        - super(str);
        - }
        - }

So jao ab bhai
Shandar bhai
Wow very dangerous