String = Class in Java
= can we modify the state of the String object after it is created }} IMMUTABLE
= toUpperCase() - it is NOT modifying "this" }} instead it will create another object that is uppercase !!
= Create string using LITERAL or new
= How is the literal string created and stored ?

```
String  s = "hello";
It is stored in CONSTANT POOL
A new object is created ONLY if same is not found in the constant pool
  String s1 = "hi";
  String s2 = "hi";
   if( s1 == s2)
        //comparing the reference   = they will be same

 String s1 = new String("hi");
String s2 = new String("hi");
        If( s1 == s2)
              //comparing the reference  = they will be different
```

Encapsulation =
Properties of the class are hidden [accessed in a controlled way]
Hidden = access specifier = private
Access in a controlled way = getters and setters

Inheritance = Relation between classes
= it is established using "extends"
= when a sub class object is created - a super class instance + sub class instance is created
= always a super() constructor is called from a sub constructor
We can pass parameters to super(x,y,) if we want to invoke
super class's parmeterized constructruct
As the first line of code
= the super.show() ; TO distinguish between super implementation and sub implementation
this.show() ; call the sub class's show

= Object class = is the Supermost class= super class of ALL classed
Even if we do not write extends still by default
Every class is a SUB class of object

Due to inheritance

```
  A  obj = new A() ;   //Valid equation RHS  = LHS
  A obj = new B();  //Valid if B extends A   i.e.  B is a type of A

//call  =     if  B extends A  , C extends B , D extends A
   f1( new A() );  //valid call
   f1( new B() ) ;   //valid call
   f1( new C() );  //valid call
   f1( new D() )  ;  //valid call

//definition
  void f1(A  obj)
  {
  }
```

Polymorphism = poly = many , morphs= forms= Implementations
Of the same function.

```
class A
{
        void f1(int x) {  ……}
        void f1(int x,int y)  {  …… }
        void f1(String x) {  ……}
}
```

Many implementations of f1 in same class = function OVERLOADING
How is one impl distinguished from other ? PARAMETER LIST

```
class A
{
        void  f1(int x){……}
}

class B extends A
{
        void f1(int x)
        {……different……}
}
class C extends B
{
        void f1(int x)
        {…..another
        }
}
```

Many implementations of f1 , PARAMETER LIST is SAME
Implementations change in the hierarchy }}} OVERRIDING

Exception Handling

All the exceptions are Subclasses of Exception class

- try = write the risky code, catch = handle the exception
- Finally block = code that runs ALWAYS
- throw = explicitly raise an exception
- throws = declare that a function may raise an exception

Package = a subfolder = folder structure where my .class file should be kept

```
package   study;
public class XYZ
{
}
```

Compiler will place the XYZ.class in the study folder.

How to access a packaged class in another packaged class ?

```
package  users;
import  study.XYZ;  //compiler should add the full name wherever XYZ is found
class  User
{
        p.s.v.main(String[] )
        {
```

```
                study.XYZ  obj= new study.XYZ();
                OR
                  XYZ  obj = new XYZ();
            }

    }
```

Class can have concrete methods or abstract methods

concrete method = the method has an implementation
abstract method = the method has no implementation
= MUST be in abstract class
= We cannot instantiate abstract class
= abstract class SHOULD BE subclassed, subclass should override abstract method

---

final class = It can never be extended by any subclass
final method = It cannot be OVERRIDDEN by any subclass

Interfaces = a construct similar to a class
= we can put it in a package
= when it compiles we get a .class file
= all functions are abstract by default
= no concrete functions
= properties can only be static and final properties
= we cannot instantiate an interface --- don't use "new"

Why are interfaces used? Polymorphism OVERRIDING
it will create a hierarchy and the subclasses will implement the abstract function

extends inheritance	implements inheritance
between two classes between two interfaces	between a class(sub) and an interface (super)

One class can extend from only one other class
One interface can extend from many interfaces
interface I1 extends I2,I3,I4
{
}
	One class can implement many interfaces
	class DesignPrint
	implements Printable, Storable, Cacheable
	{
	}

IS A relation , Hierarchy	IS A relation Hierarchy is created

```
interface  One
{
        void f1() ;
        void f3();
}

 class Impl  implements One,Two
{
            void f1(){  ……sout("ABC") ;   }
            void f2() { ……}
```

Interface Two { void f1() ; void f2(); }

```
            void  f3(){ ….}
 }

  class User
    main
            One  obje = new Impl();
              obje.f1() ;   ----ABC
              obje.f3();

            Two  obj = new Impl();
              obj.f1() ;  ---------ABC
```

File IO in Java

String, System , Integer} default imported package **java.lang**

IO = InputStream and OutputStream classes in **java.io** package

InputStream

Stream = flow of bytes

InputStream = Flow of bytes from INPUT DEVICE (KBD, FILE)
To **BUFFER IN RAM<==>Variable in JVM**
OutputStream = Flow of bytes from **BUFFER IN RAM<==>Variable in JVM**
To OUTPUT DEVICE (MONITOR, FILE)

FileInputStream extends InputStream
FileOutputStream extends OutputStream

A. Write to a Text file .
B. Read from the Text file

Two types of Files

1. Text file = use Text encoding formats like ASCII or UNICODE
   Hi 12 Hispace 1 2 all are stored in ASCII
   Everything that opens in a notepad is a text file
   .java , .txt, .xml
2. Binary file = use formats for every data type
   Hi 12 Hispace are stored in ASCII , 12 will be stored in integer format
   Binary files need their own readers and writers
   Docx . Ppt , xsl , .class , .gif , .wav

Java--Variable =====> PrintWriter =====> FileOutputstream ===> File Buffer==> File

KBF ========>Scanner =======> sc.nextLine()====> Java-Variable ====>sout

Read from the File and Write on Monitor
A .BufferedReader

FILE=====>FileReader===>BufferedReader ====> readLine() ===> String java variable

B. Scanner

Serialization = saving object to a binary file or sending object on a network stream

Where is the object created ? Heap ===>JVM ===>RAM (volatile memory)
|
|
Object Persistence = save the object even after power off

Pillars of OOP
1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

minor pillars
5. Persistence
6. Strong typing
7. Concurrency

Saving object to a File ==>HardDisk (non volatile) = Object persistence
Technique of saving object to a file = Serialization
the object with all its properties and inherited properties , contained properties
+ info about the classes and types
Everything is written to file in a particular format }} SERIALIZATION

RAM =====object======> HDD (serialization)

RAM <======object======HDD (Deserialization)

1. class whose object can be serialized
2. ObjectOutputStream = Serialize
3. ObjectInputStream = Deserialize

SerializeEx =========OBJ ==============>books.pra
|
|
obj
|
DeSerializeEx

Multithreading = Concurrency Management
Thread = Path of execution within a process
Process = JVM = Program in execution =
gets a separate process space [class area , stack area, heap area] in RAM
and processId
THREAD ----------
Every path has a start point
sequence of steps
sequence of instructions = statements

Every path has an end point

sequence of instructions = statements

Every path has an end point

THREAD is compulsary in a process = as it will tell the sequence of instruction execution

EVERY process has a DEFAULT thread = main thread

```
public static void main(String[] ) {      begin of path(thread)
        Statement1
        Statement2
} //end of main     end of path (thread)
```

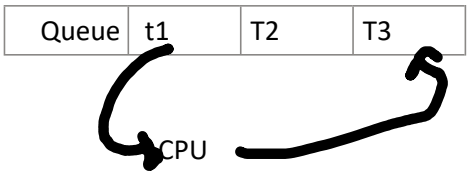
Multithreading =

WE have Many paths of execution that do not wait for each other to complete

These paths= threads run in ROUND ROBIN scheduling

Every thread gets a time slice to run , after that it waits in ready queue, resumes on its turn

Other thread will not wait for earlier thread to complete its code



Default thread = public static void main(String [])

Alternate threads = public void run() { }

Many threads within the same process

Each thread shares the class area and heap area of the process

However each thread will get a separate stack!!!!!

Threads may share data !!!

The shared data may lead to data corruption = RACE CONDITION

The **Thread Safety** means preventing RACE CONDITION

synchronized

Mutual Exclusion of the code[CRITICAL SECTION] that uses shared data

Th1 is running code that uses shared data[CRITICAL SECTION]

Then even if Th1 takes a break

Th2 cannot run its code that uses shared data[CRITICAL SECTION]

synchronized will help programmer specify [CRITICAL SECTION]

```
synchronized(shareddata)
    {
      code that uses shared data
    }

     synchronized  void f1()
     {
          Code that should run exclusively
     }
```

Interfaces -

Interface with 0 abstract functions = **Marker** Interface /Tag interface

Example - java.io.Serializable

Interface with 1 abstract function EXACTLY = Functional Interface
Example - java.lang.Runnable

Interfaces with more abstract functions!!!

If we are having a Functional Interface
It can be implemented with a SHORTHAND notation
LAMBDA Notation

Generics = Using a class for different data types without changing the .class
= Also used to avoid ClassCastException

Create a Stack
- array of String
- constructor
-push
-pop
-display

If I want a Stack of Book?
If I want a Stack of Toy ?

Collections In Java = The java implementation of the Data Structures

Collection interface = API for working on a collection
Guest List - on Pen and Paper
Operations that can be performed on Guest List ----
1. append
2. insert into the list
3. remove from the list
4. add another list to this list
5. traverse the list = visit each element
6. count
7. sorting
8. searching

Collection interface = **abstract** functions for **common operations** on a collection of element
Interface List **extends** Collection }} List is a Collection
- Growable collection
- Allows Index Based Access
- Allows duplicate entries

Class ArrayList **implements** List
class LinkedList implements List

Will the implementation of abstract public void add(element) be same for ArrayList & LinkedList ???

---

interface Set **extends** Collection
---- it is not using duplicate entries = all entries are unique
---- it is not using indices = index based access
---- Growable

class TreeSet implements Set }} It gives a Binary Tree representation
class HashSet implements Set
class LinkedHashSet implements Set
