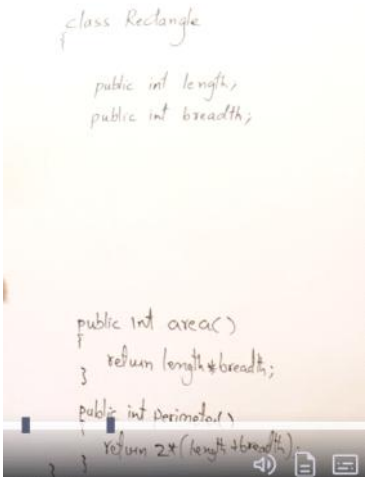
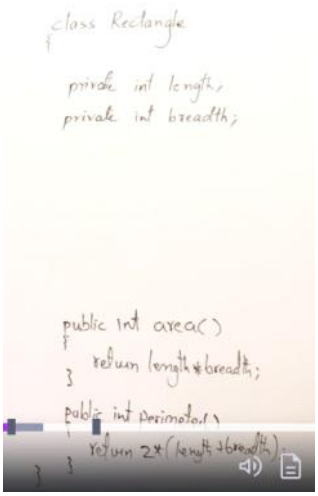
Java 5.0

Tuesday, January 27, 2026 12:43 PM

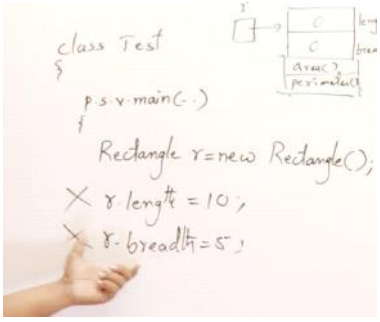
The engine (data) is hidden for simplicity and only operations can allow us drive the car
So data is hidden and operations are made visible.
Users should enjoy using the product

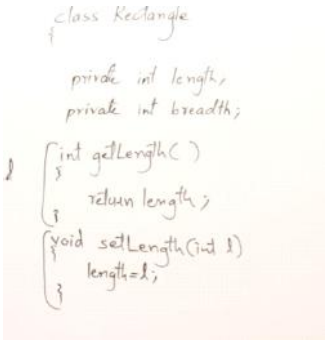


Now we makin data private.

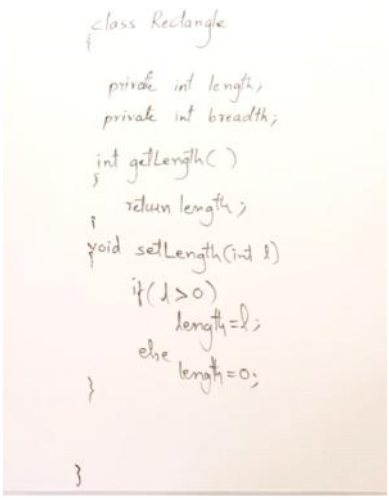


Now we can't access data just by creating the object of the rectangle class

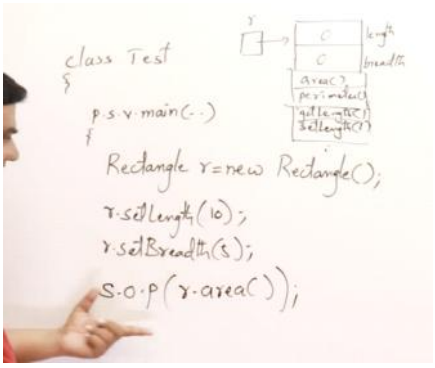




We used validatiion in setter below:

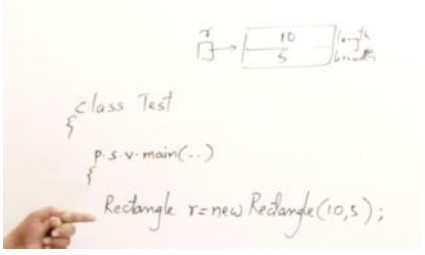


Main()

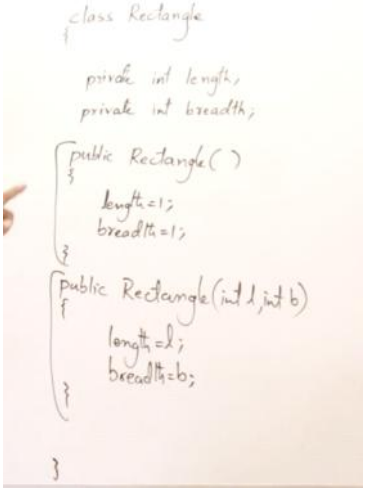


Constructors:

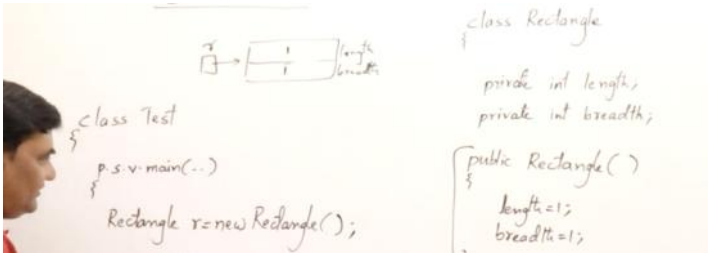
Look, in the previous data hiding examples, the length and breadth were 0 - but if we look in real life, data have some value by default. We didn't write the constructor there, but default constructor was there by default- created automatically when object is created.



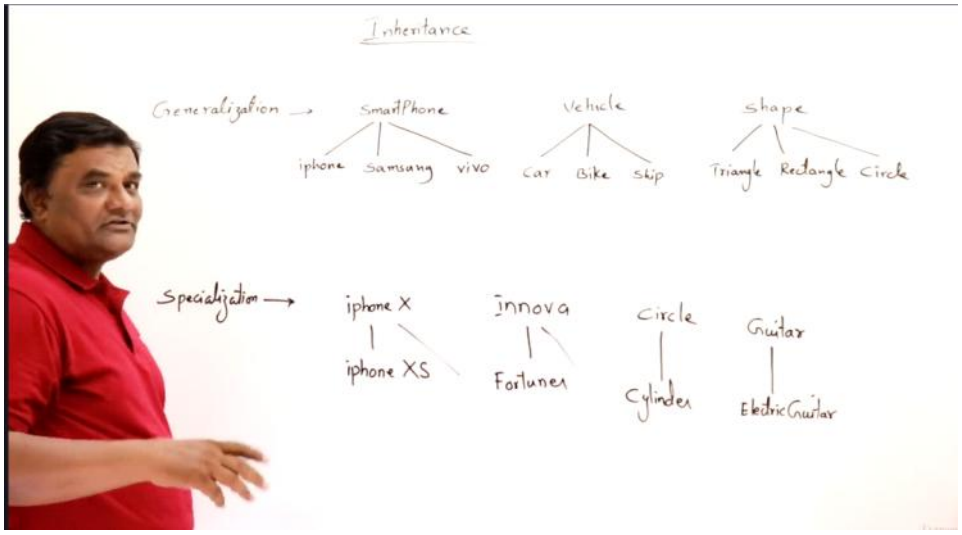
Now we writing our own constructor



An example of default constructor, but setting them to 1 instead of 0, and not using setter and getter.

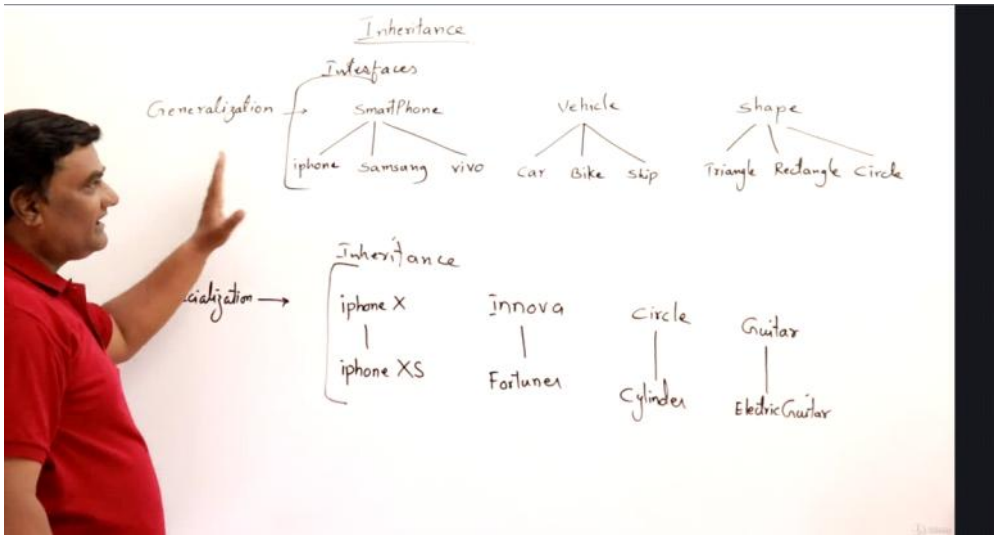


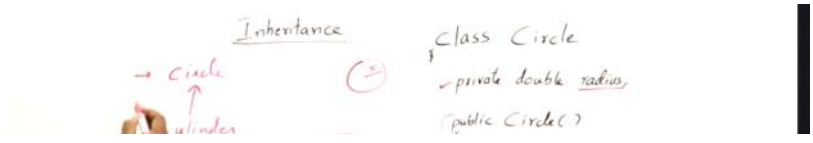
Inheritance

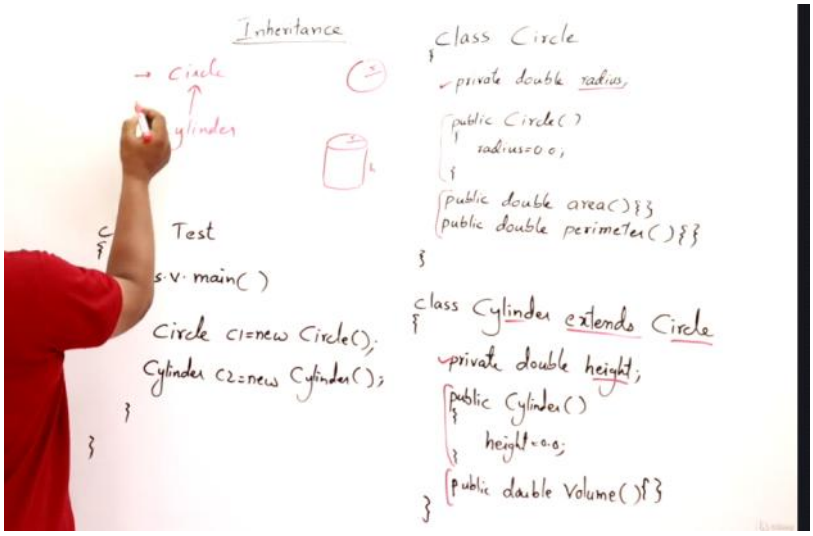


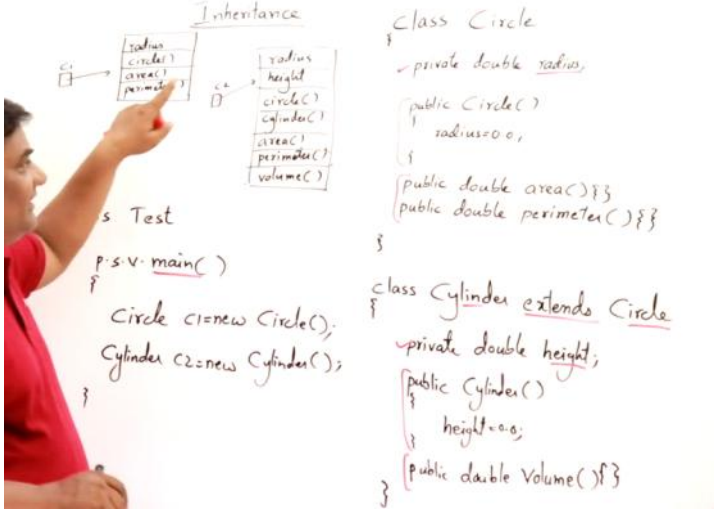
Generalization -> bottom up
Specialization -> top down

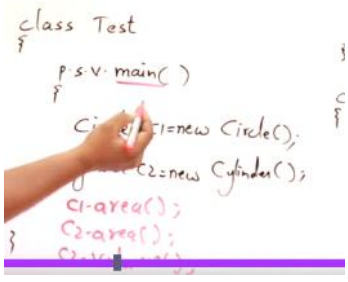
One is achieved through inheritance, other is achieved through interfaces. Something in between these 2 -> abstract classes











Constructors in Inheritance

```java
package inheritconst;

class Parent
{
    public Parent()
    {
        System.out.println("Parent Constrcutor");
    }

}

class Child extends Parent
{
    public Child()
    {
        System.out.println("Child Constructor");
    }
}

public class InheritConst
{

    public static void main(String[] args)
    {

    }

}
```

```java
package inheritconst;

class Parent
{
    public Parent()
    {
        System.out.println("Parent Constrcutor");
    }

}

class Child extends Parent
{
    public Child()
    {
        System.out.println("Child Constructor");
    }
}

public class InheritConst
{

    public static void main(String[] args)
    {
        Child c=new Child();
    }

}
```

Ok, now when Child object is creted, which constructor will be run? Parent or Child?

Output:

```
Deleting: /Users/abdulbari/NetBeansProjects/InheritConst/build/buil
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/InheritCo
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/Inherit
compile:
run:
Parent Constrcutor
Child Constructor
BUILD SUCCESSFUL (total time: 0 seconds)
```

when Child object is creted, parent constructor is executed first, then child constructor.

Parameterized Constructor

```java
package superconst;

class Parent
{
    Parent()
    {
        System.out.println("Non-Param of Parent");
    }
    Parent(int x)
    {
        System.out.println("Param of Parent");
    }
```

```java
class Child extends Parent
{
    Child()
    {
        System.out.println("Non-Param of Child");
    }
    Child(int y)
    {
        System.out.println("Param of Child");
    }
}
```

```java
public class SuperConst
{
    public static void main(String[] args)
    {
        Child c=new Child();
    }
```

1:45 / 10:24

Output:

```
compile:
run:
Non-Param of Parent
Non-Param of Child
BUILD SUCCESSFUL (total time: 0 seco
```

1:46 / 10:24

Now we pass parameters on Child c object.

```java
public class SuperConst
{
    public static void main(String[] args)
    {
        Child c=new Child(20);
    }
```

2:16 / 10:24

Now non-param of parent will be called followed by param of child.

Output:

```
Non-Param of Parent
Param of Child
BUILD SUCCESSFUL (tota
```

well after running this java code,
We get output:
Non-Param of Parent
Param of Child

But why didn't we get output Param of Parent as well? pls say lucidly

Short answer:

Because you created the object with `new Child(20)`, Java ran the **parameterized constructor of `Child`**. Since that constructor doesn't explicitly call `super(...)`, the compiler automatically inserts **`super()`** as the very first statement. That calls the **no-arg constructor of `Parent`**, not `Parent(int)`. Hence you see:

```
Non-Param of Parent
Param of Child
```

—not "Param of Parent".

```java
class Parent {
    Parent() {
        System.out.println("Non-Param of Parent");
    }
    Parent(int x) {
        System.out.println("Param of Parent");
    }
}

class Child extends Parent {
    Child() {
        System.out.println("Non-Param of Child");
    }
    Child(int y) {
        System.out.println("Param of Child");
    }
}
```

And in `main`:

```java
Child c = new Child(20);
```

Rules to remember:

1. **Every constructor must call a super/this constructor as its first statement.**
   - If you don't write one, Java inserts **super()** automatically.
2. **super() calls the parent's no-arg constructor.**
   super(x) would call the parent's parameterized constructor.
3. You can't call both this(...) and super(...) in the same constructor—they must be the **first** statement, and there can be **only one** first statement.

Applying the rules:

- Child(int y) doesn't start with super(...) or this(...).
  → The compiler inserts super();
  → This calls Parent() (no-arg), printing **"Non-Param of Parent"**.
- Then the body of Child(int y) prints **"Param of Child"**.

How to get "Param of Parent" instead

Explicitly call the parent's parameterized constructor from the child's parameterized constructor:

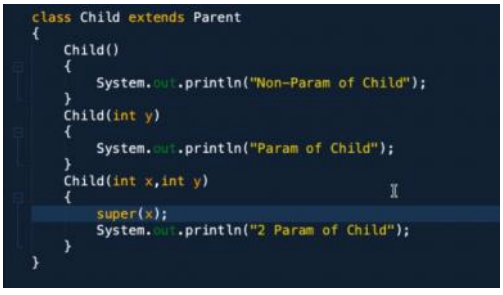
```
class Child extends Parent {
    Child(int y) {
        super(y);  // <-- explicit call to Parent(int)
        System.out.println("Param of Child");
    }
}
```

Now new `Child(20)` will print:

```
Param of Parent
Param of Child
```

What will be executed as object c is made?

```
class Child extends Parent
{
    Child()
    {
        System.out.println("Non-Param of Child");
    }
    Child(int y)
    {
        System.out.println("Param of Child");
    }
    Child(int x,int y)
    {
        System.out.println("2 Param of Child");
    }
}

public class SuperConst
{
    public static void main(String[] args)
    {
        Child c=new Child(20);
    }

}
```



```
class Child extends Parent
{
    Child()
    {
        System.out.println("Non-Param of Child");
    }
    Child(int y)
    {
        System.out.println("Param of Child");
    }
    Child(int x,int y)
    {
        super(x);
        System.out.println("2 Param of Child");
    }
}
```

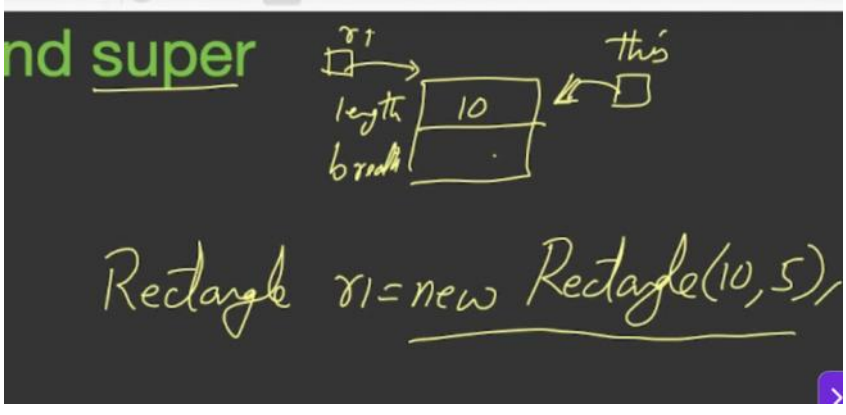
By super(x), we are calling the Parent(x) constructor

Incase Child class contains 2 parameters

```
    }
    Parent(int x)
    {
        System.out.println("Param of Parent");
    }
}

class Child extends Parent
{
    Child()
    {
        System.out.println("Non-Param of Child");
    }
    Child(int y)
    {
        System.out.println("Param of Child");
    }
    Child(int x,int y)
    {
        super(x);
        System.out.println("2 Param of Child");
    }
}

public class SuperConst
{
    public static void main(String[] args)
```

this and super

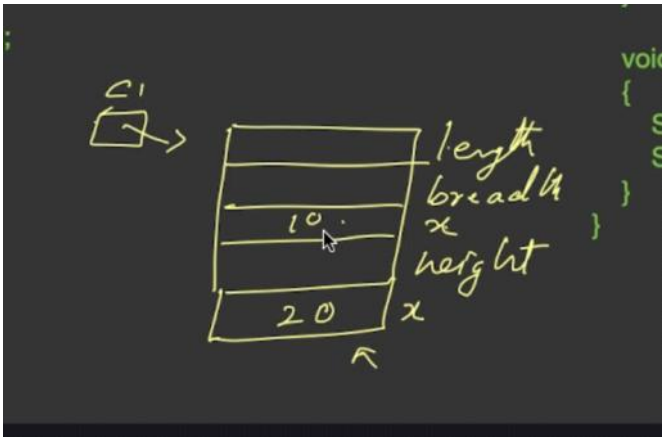
```
class Rectangle
{
   int length;
   int breadth;

   Rectangle(int l,int b)
   {
      this.length=l;
      this.breadth=b;
   }

   void display()
   {
      System.out.println("Length :"+this.length);

      System.out.println("Breadth :"+this.breadth);
   }
}
```

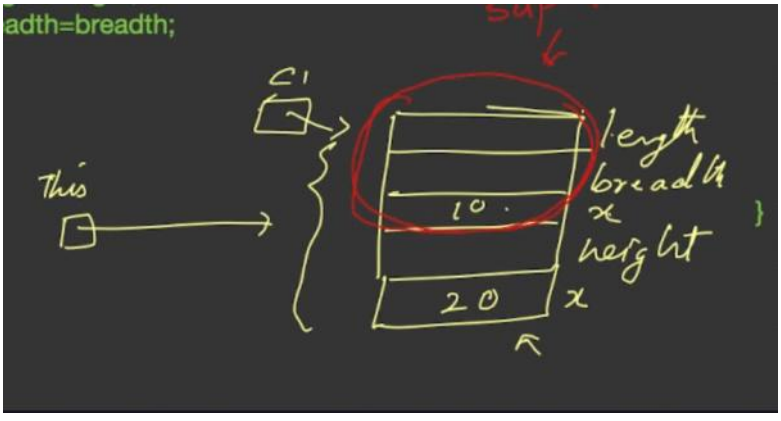


this and super

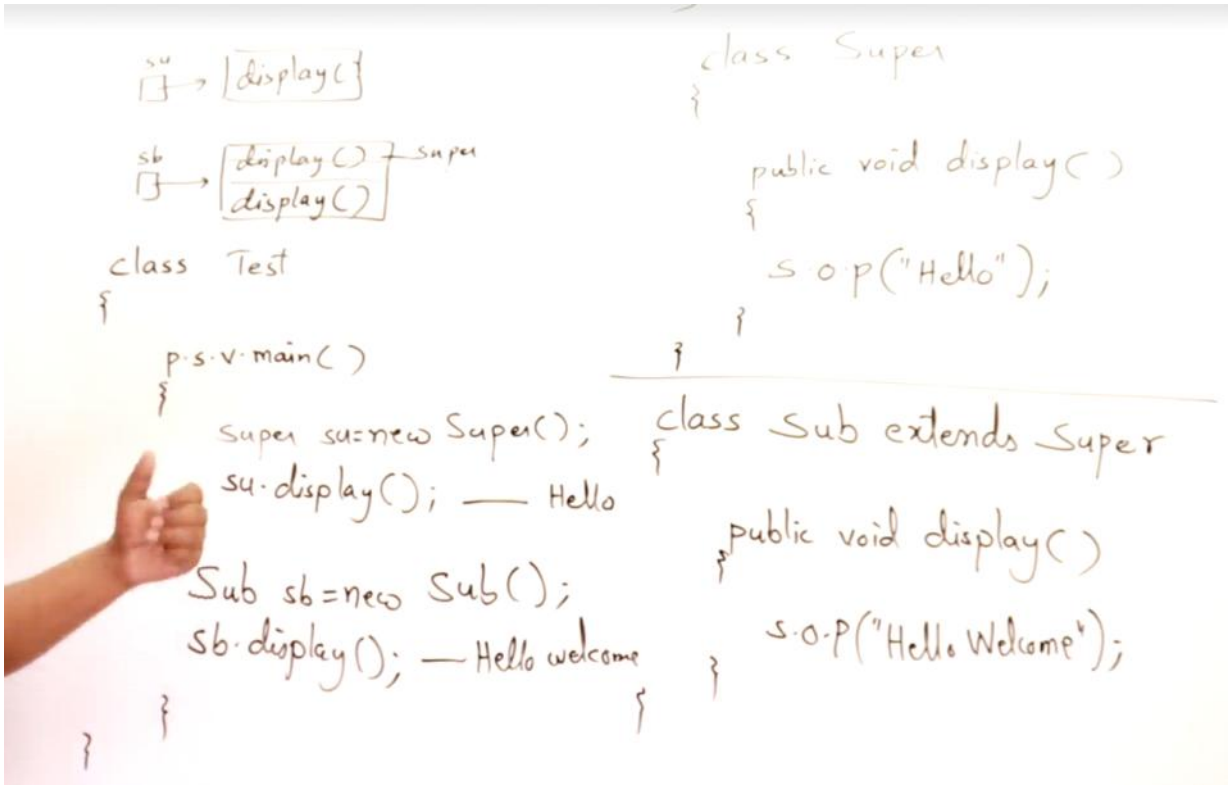
```
class Rectangle
{
   int length;
   int breadth;
   int x=10;

   Rectangle(int length,int breadth)
   {
      this.length=length;
      this.breadth=breadth;
   }
}
```

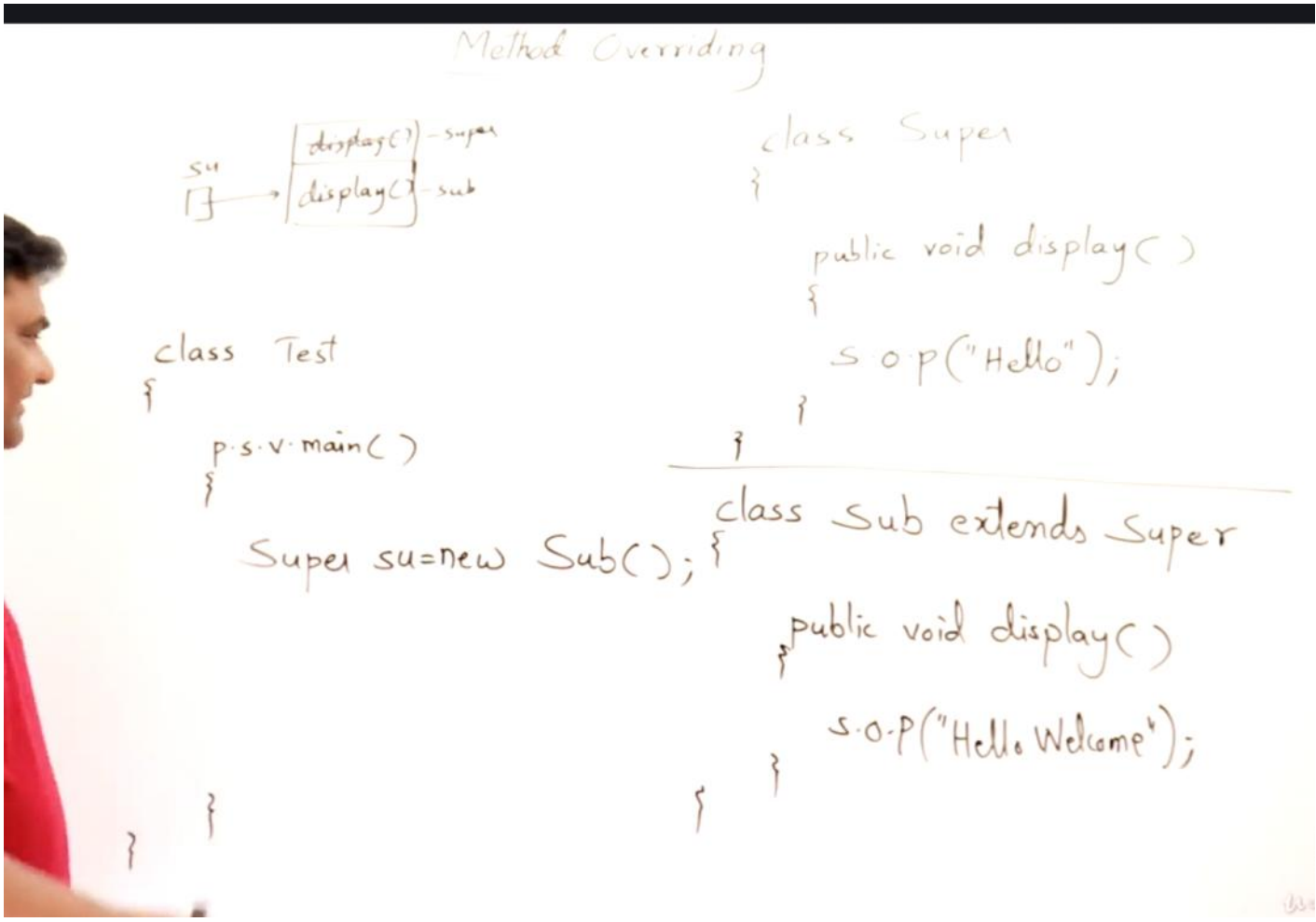
```
class Cuboid extends Rectangle
{
   int height;
   int x=20;

   Cuboid(int l,int b,int h)
   {
      super(l,b);
      height=h;
   }

   void display()
   {
      System.out.println(super.x);
      System.out.println(x);
   }
}
```

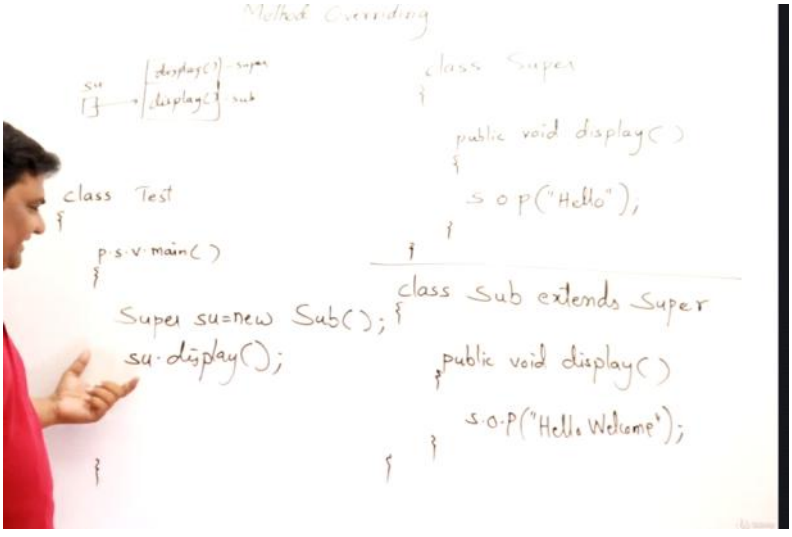




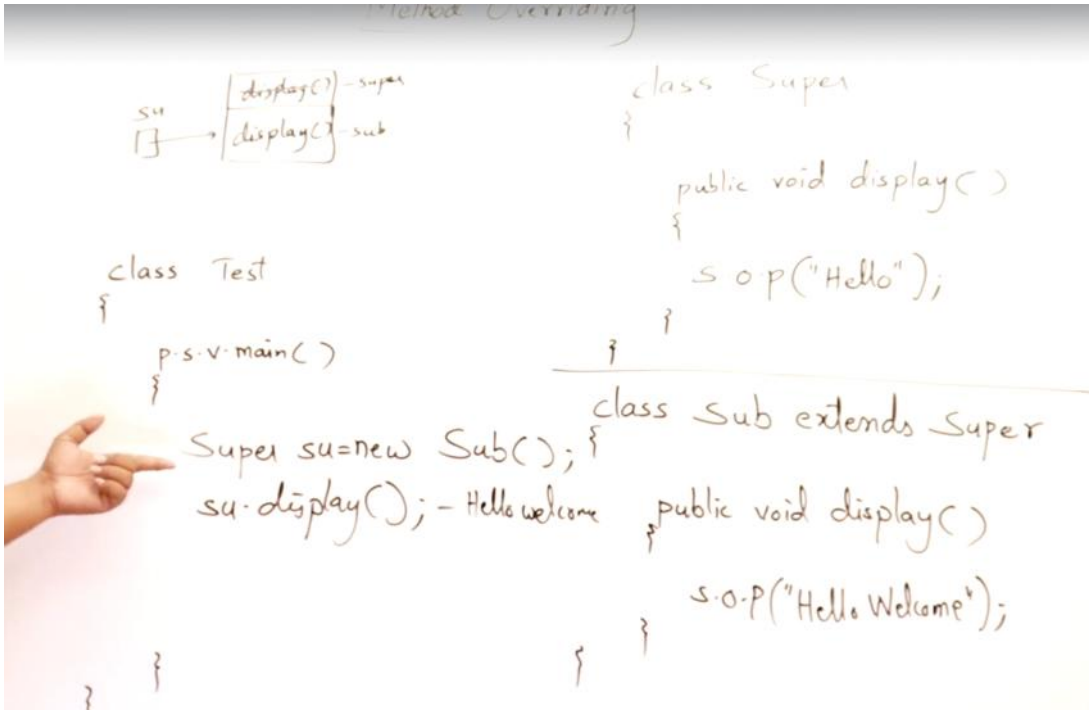
Overriding


su
display()
sb
display()
display()
super
class Test
{
p.s.v.main()
{
Super su=new Super();
su.display(); — Hello
Sub sb=new Sub();
sb.display(); — Hello welcome
}
}
class Super
{
public void display()
{
s.o.p("Hello");
}
}
class Sub extends Super
{
public void display()
{
s.o.p("Hello Welcome");
}
}


Method Overriding
su
display() - super
display() - sub
class Super
{
public void display()
{
s.o.p("Hello");
}
}
class Test
{
p.s.v.main()
{
Super su=new Sub();
}
}
class Sub extends Super
{
public void display()
{
s.o.p("Hello Welcome");
}
}


Method Overriding
su
display() - super
display() - sub
class Test
{
p.s.v.main()
{
Super su=new Sub();
su.display();
}
}
class Super
{
public void display()
{
s.o.p("Hello");
}
}
class Sub extends Super
{
public void display()
{
s.o.p("Hello Welcome");
}
}

Which display() will be called here??? Will it be called depending upon the reference (Super), or depending upon the object (Sub)??
Ans - The method will be called depending upon the object, not upon the reference.

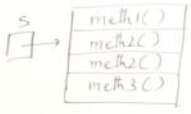


Dynamic method dispatch is sued for runtime polymorphism

```
class Super
{
    void meth1(){s.o.p("meth1");}
→   void meth2()
    {
        s.o.p("Super meth2");
    }
}
```

```
class Sub extends Super
{
→   void meth2()
    {
        s.o.p("Sub meth2");
    }

    void meth3(){s.o.p("meth3");}
}
```

Dynamic Method Dispatch



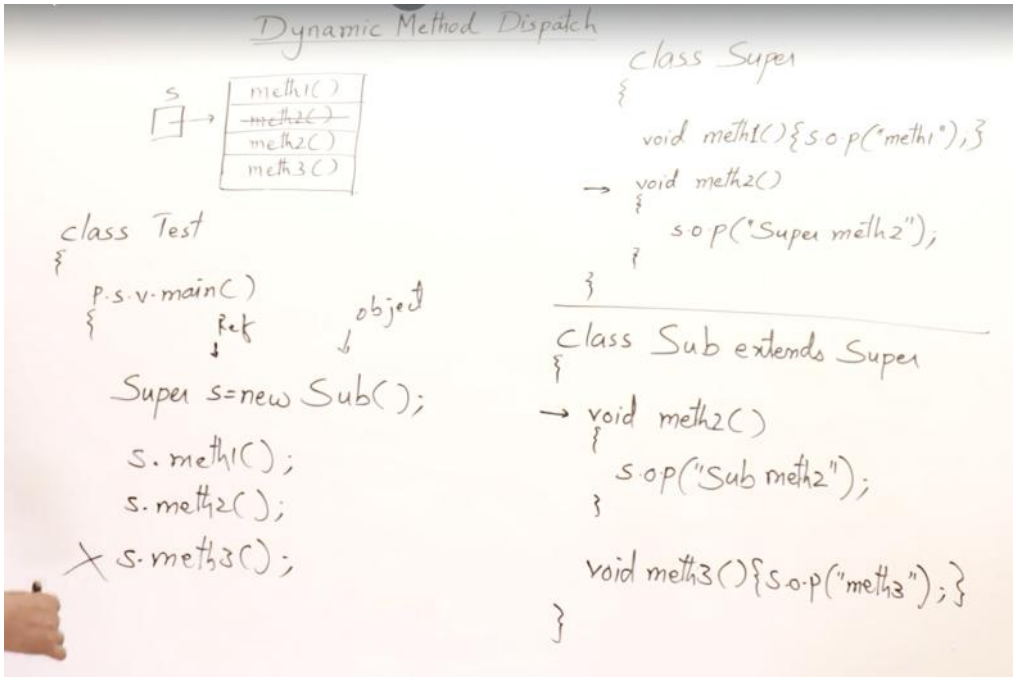

```
class Test
{
    p.s.v.main()
    {
        Super s=new Sub();
        Sub su=new Super();
    }
}
```

ref

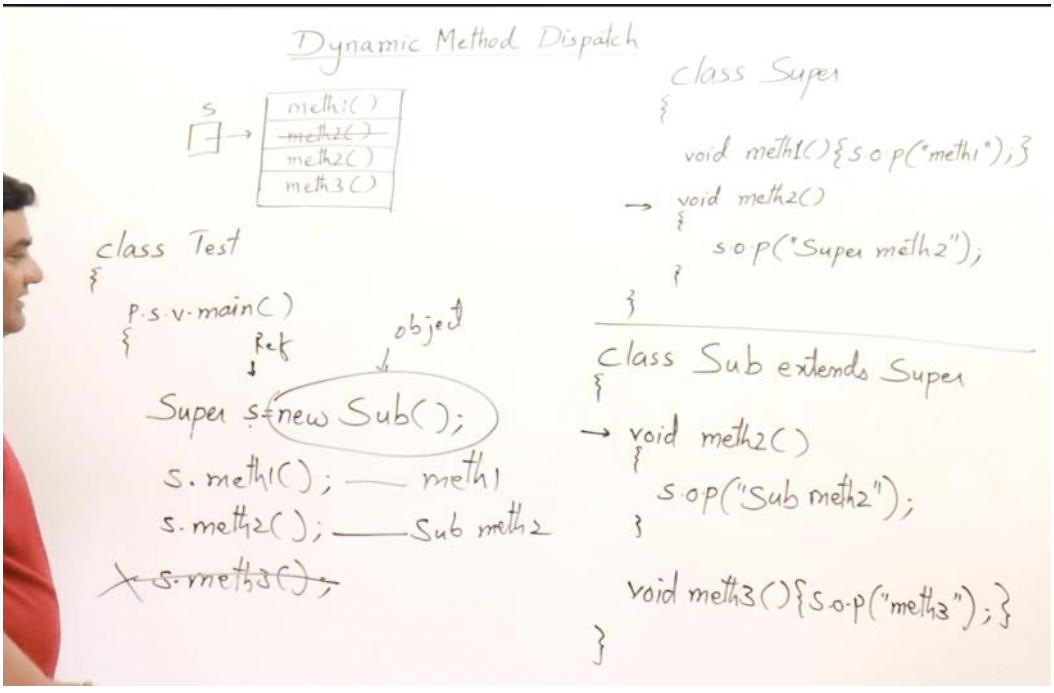
object

```
class Super
{
    void meth1(){s.o.p("meth1");}
→   void meth2()
    {
        s.o.p("Super meth2");
    }
}
```

```
class Sub extends Super
{
→   void meth2()
    {
        s.o.p("Sub meth2");
    }

    void meth3(){s.o.p("meth3");}
}
```

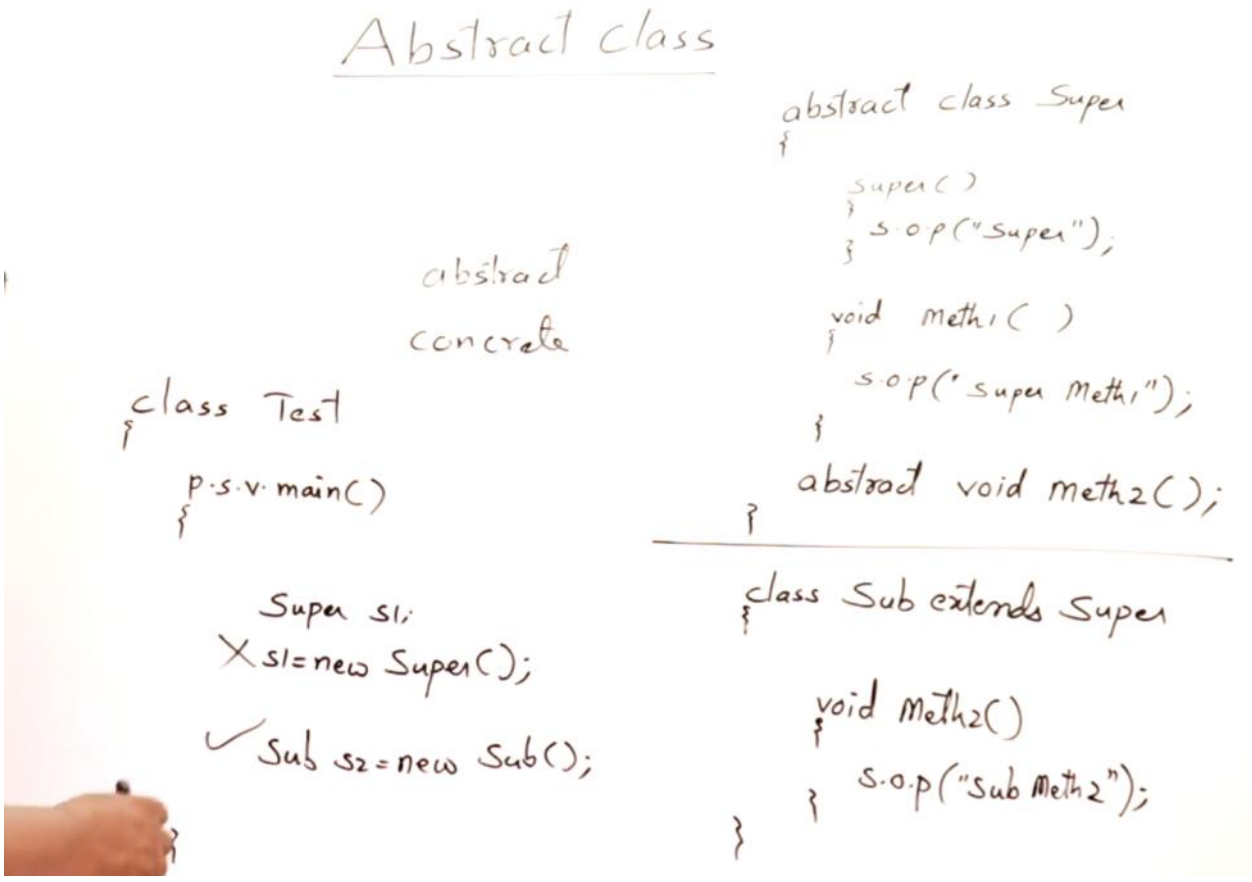


We cant call meth3 because the reference is of superclass

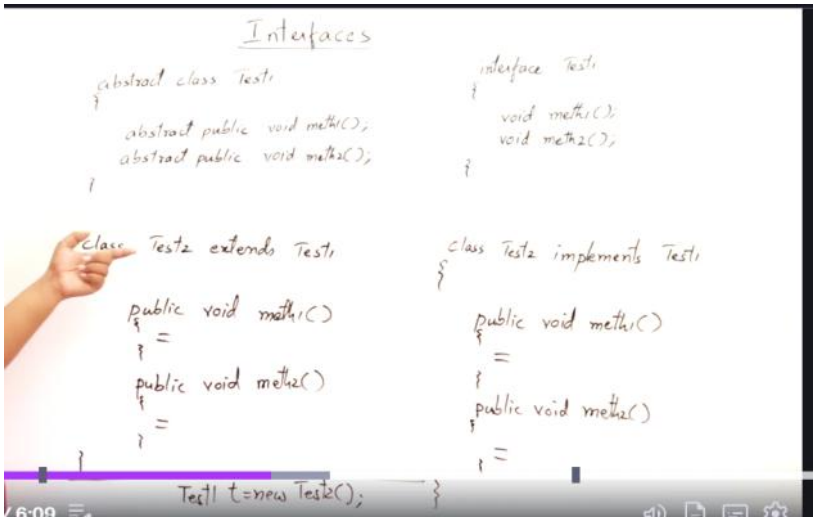


You cant create objects of an abstract class. Can create reference of it tho.
Abstract method - incomplete method

Abstact class - at least one abstract method
If any class is inheriting from abstract class, that class also becomes abstract, but if that overrides all methods then it become concrete.



Interfaces

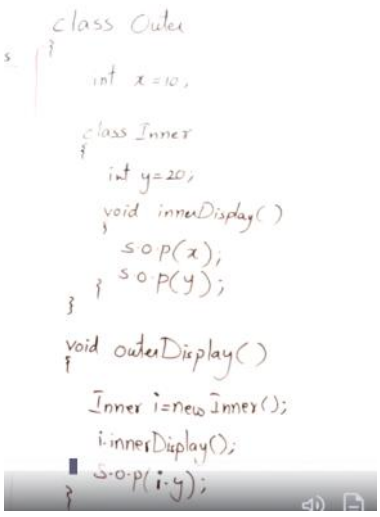


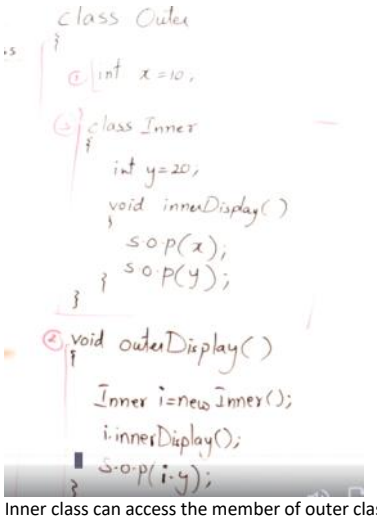
Inner Classes

Inner class

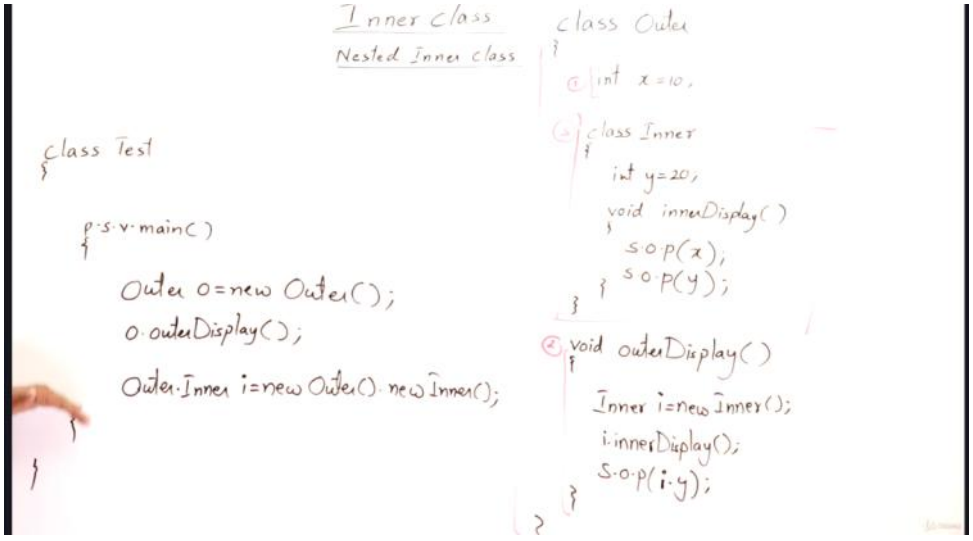
1. Nested Inner class
2. Local inner class
3. Anonymous inner class
4. Static inner class

Example

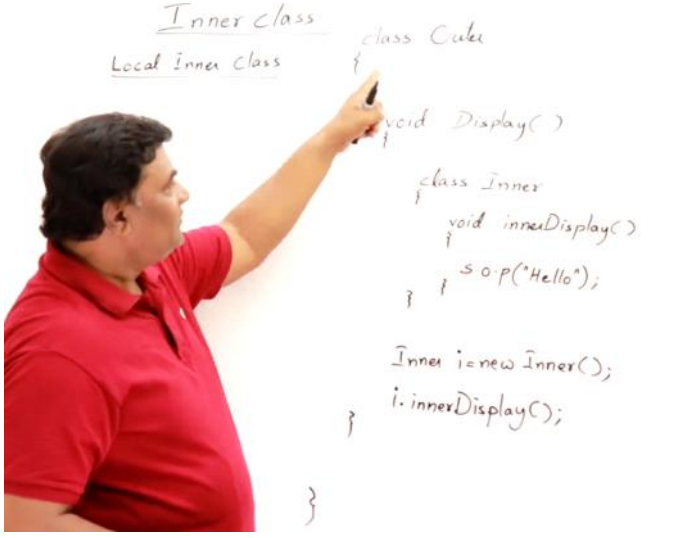


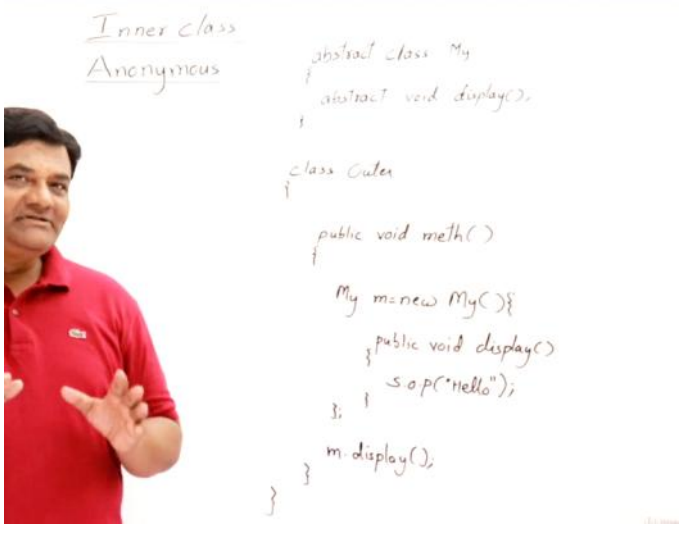


Inner class can access the member of outer class directly whereas outer class cannot Access the member of inner class directly.

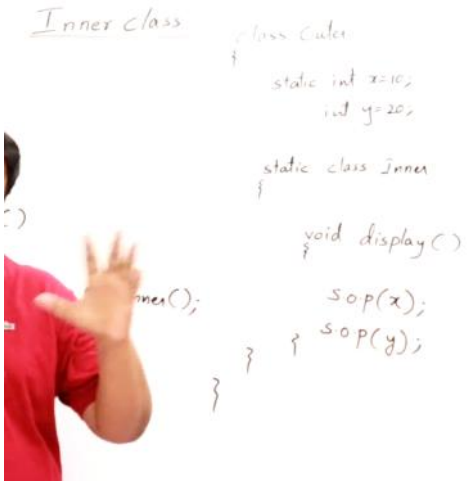


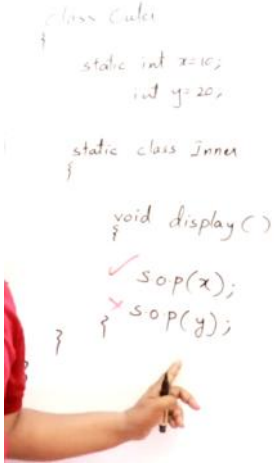
Local inner class

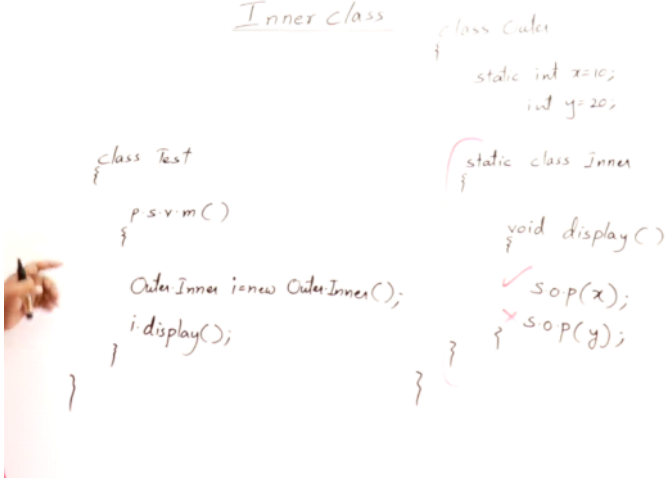




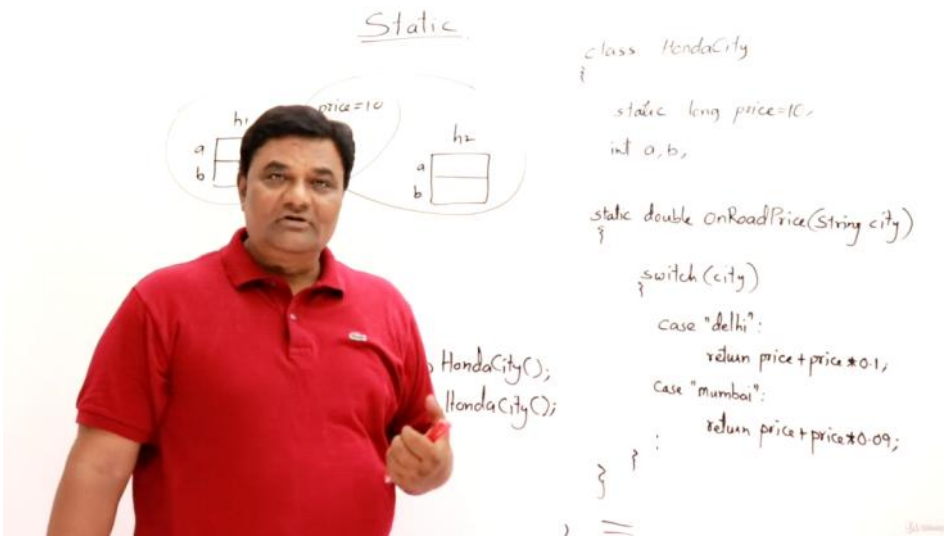
Static inner class

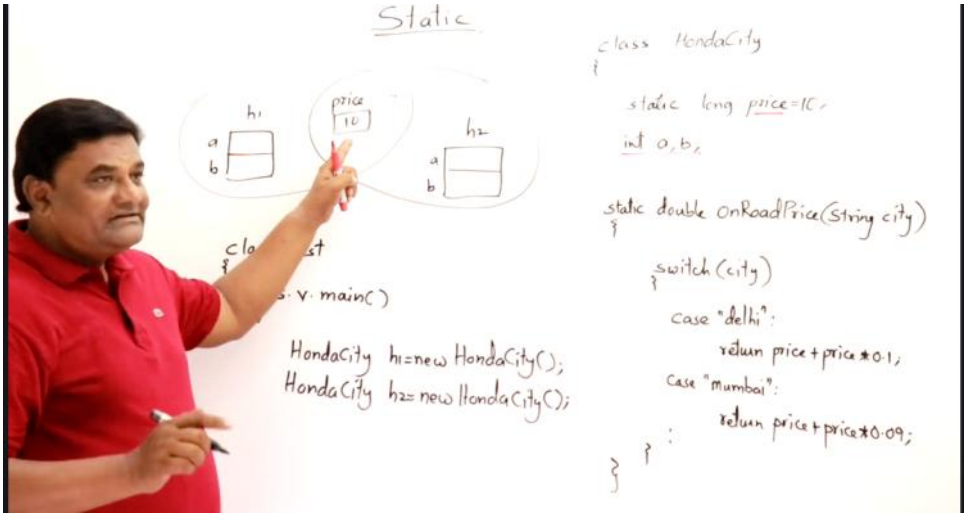


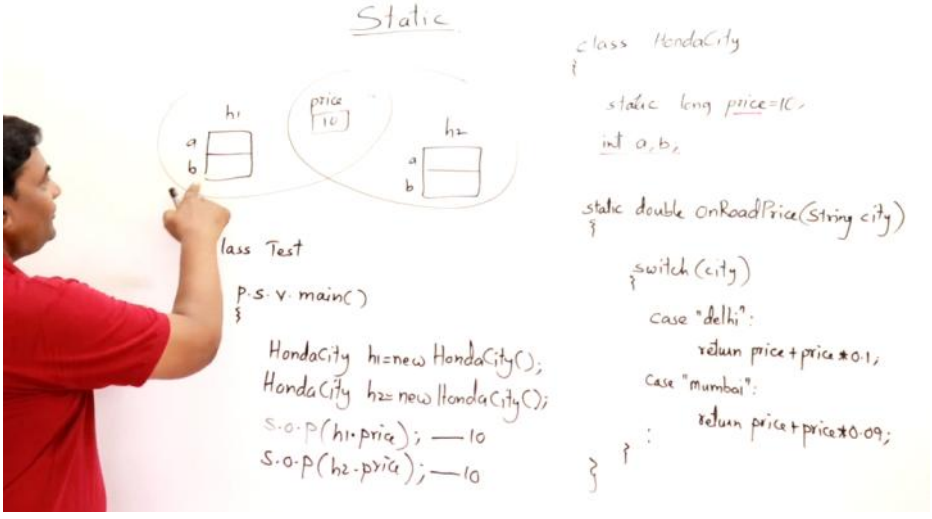




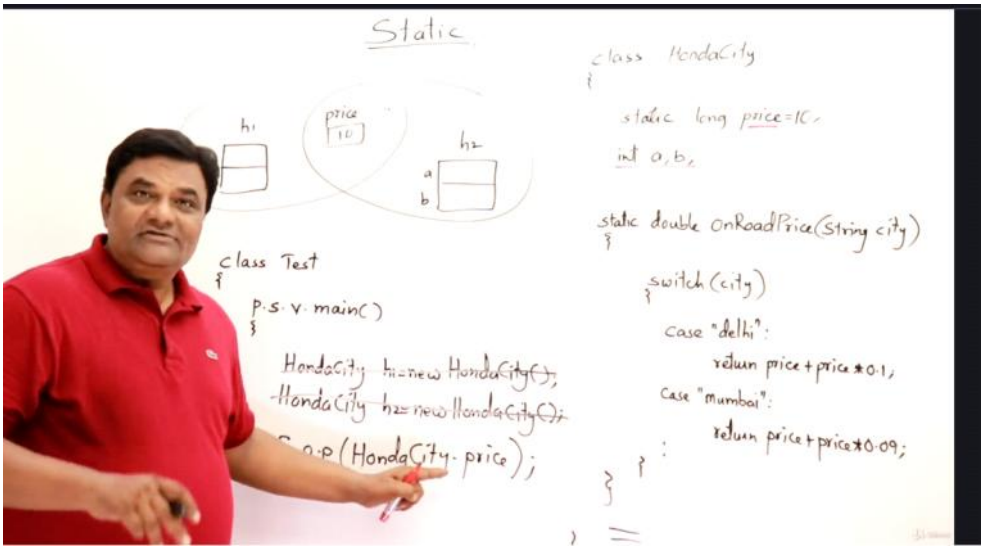
Static Members



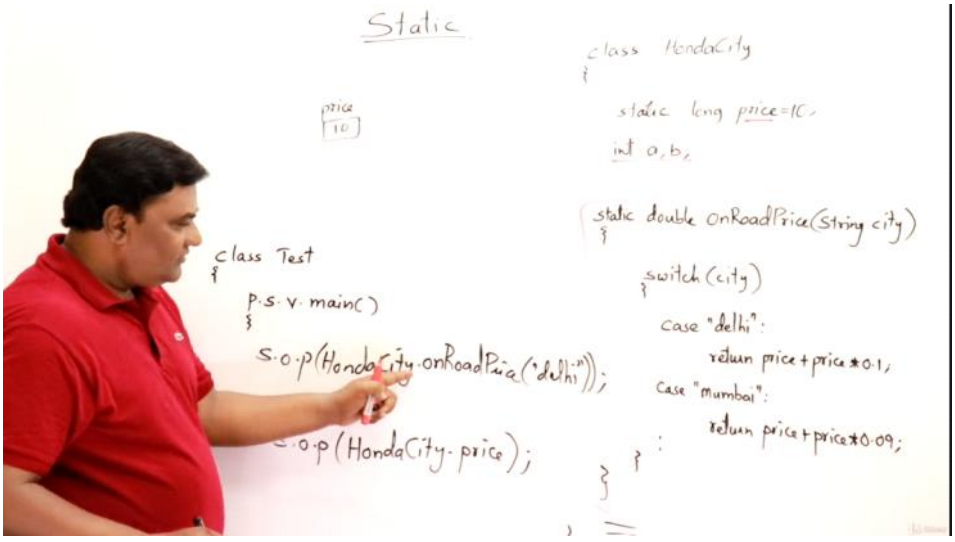




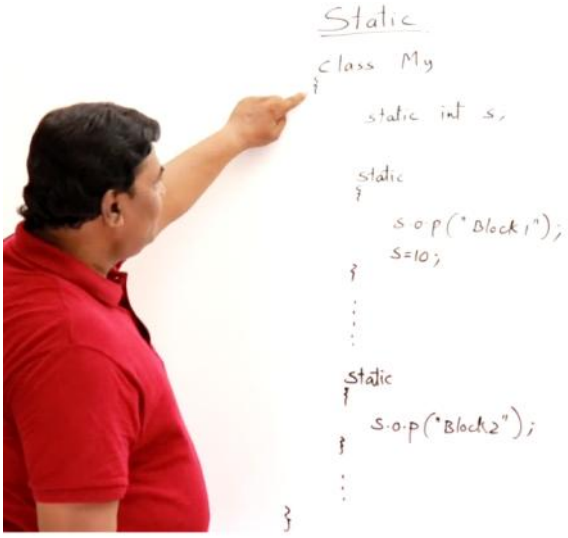
Static -> shared data
Static variables can be accessed by only using class name.



Static methods can only access static members, non-static members can't be accessed by static methods.
onRoadPrice can't access a and b.



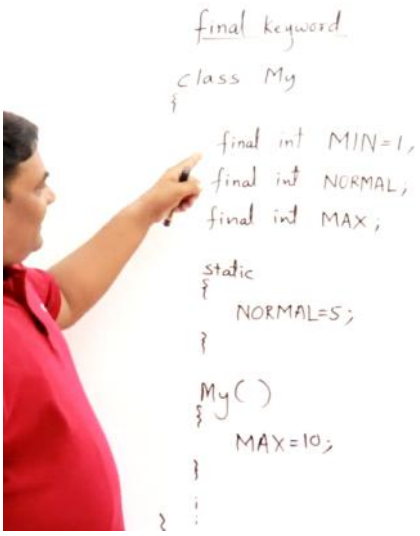
Static blocks



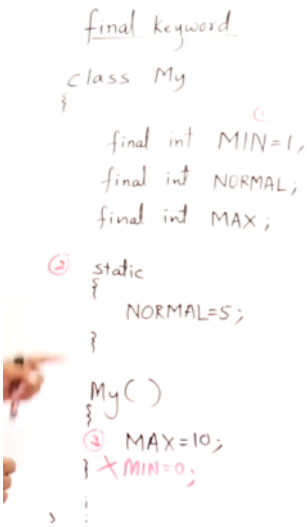
FinAL

final keyword

1 final variable
2 final Method
3 final class

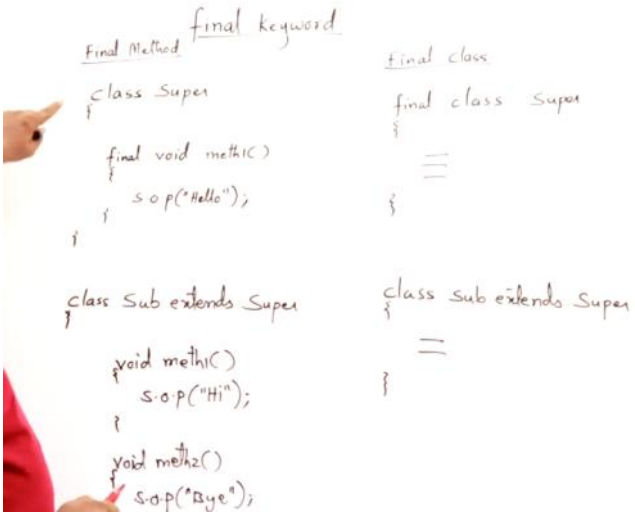


3 ways of initialising final variables - in place, blocks and constructor.
If constructor, in every constructor it must be initialised.

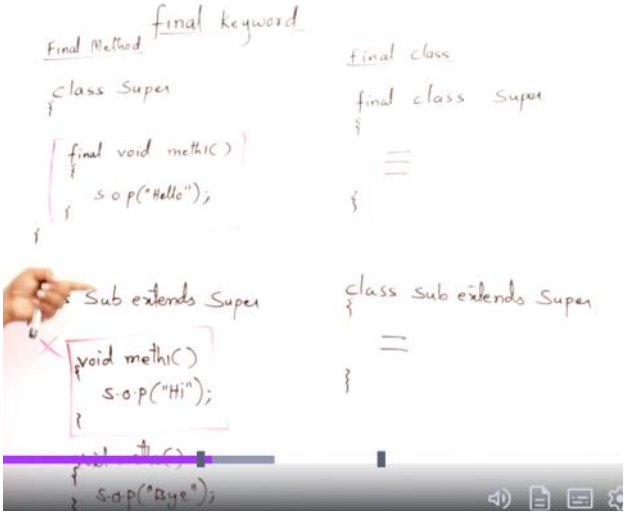


Final variable can't be modified.

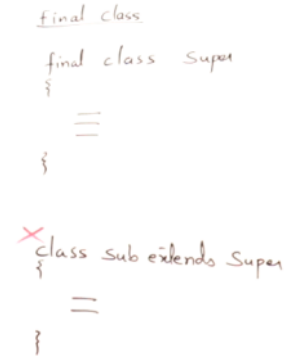
Final Method:



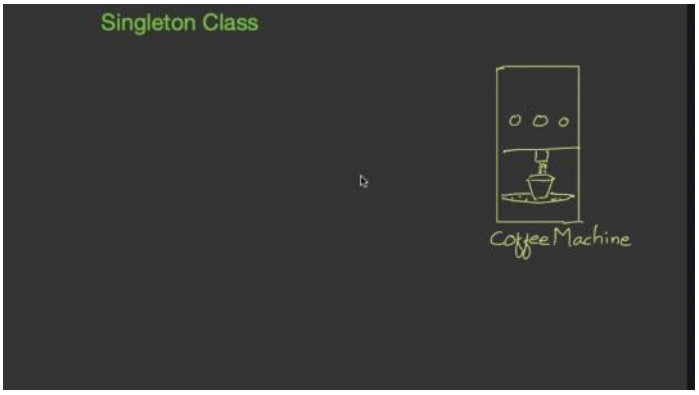
Final methods cant be overridden.

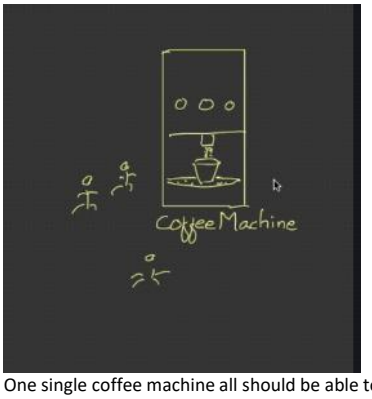


Class as Final:

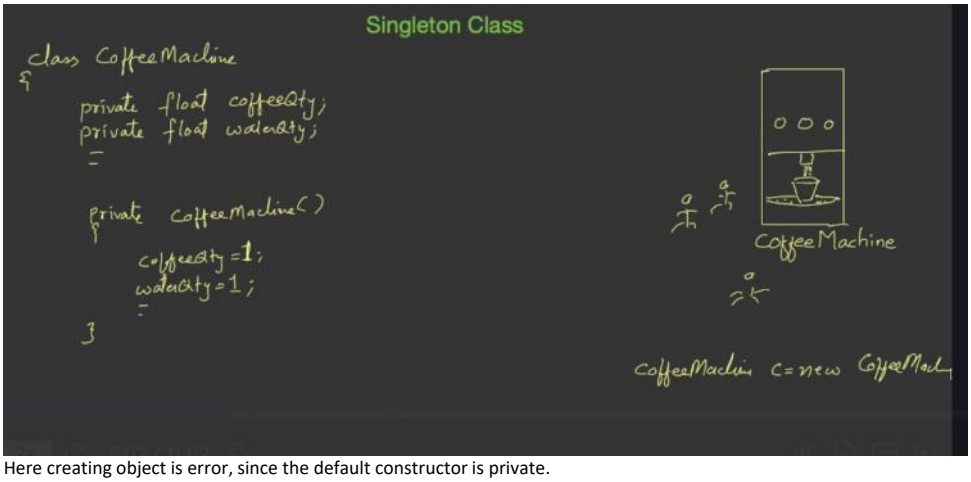


Singleton Class

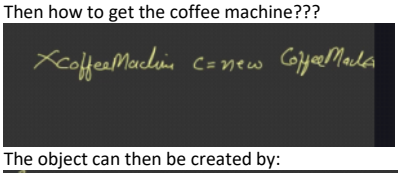




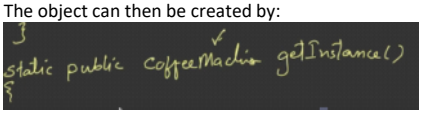
One single coffee machine all should be able to use it. How to write such a class.

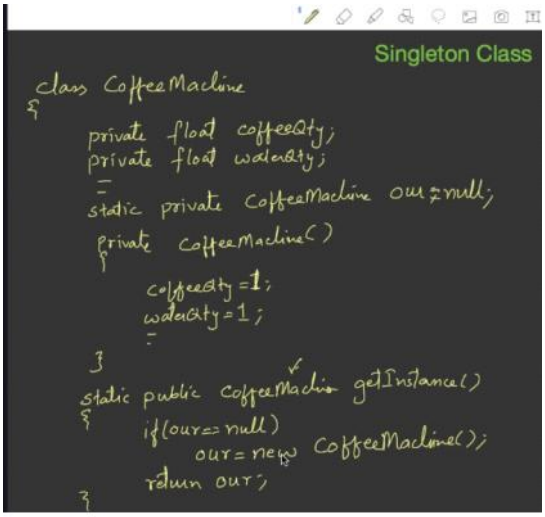


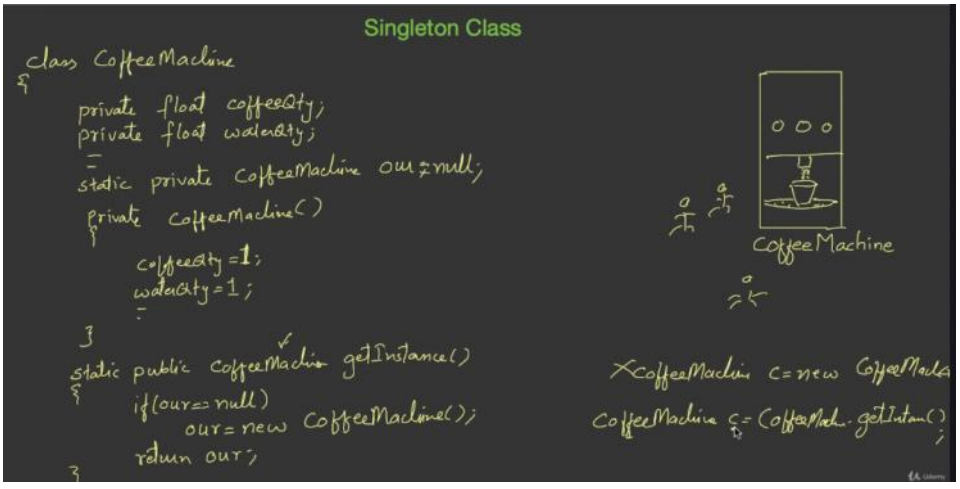
Here creating object is error, since the default constructor is private.
Then how to get the coffee machine???



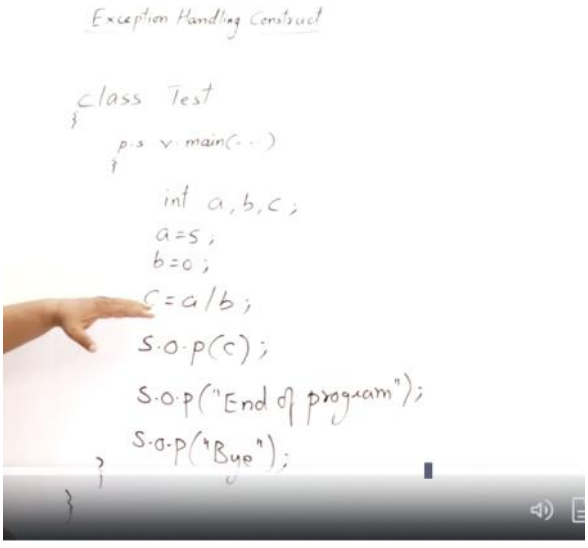
The object can then be created by:

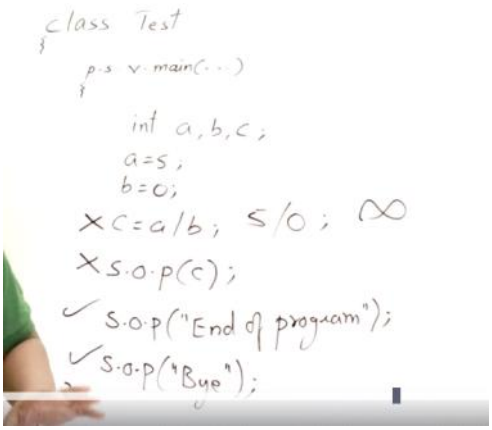




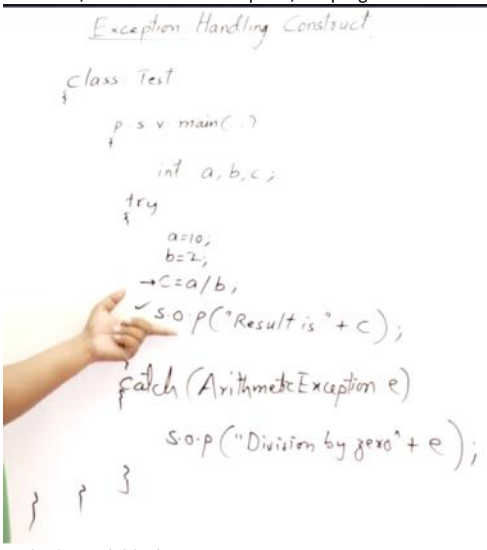


Exception handling

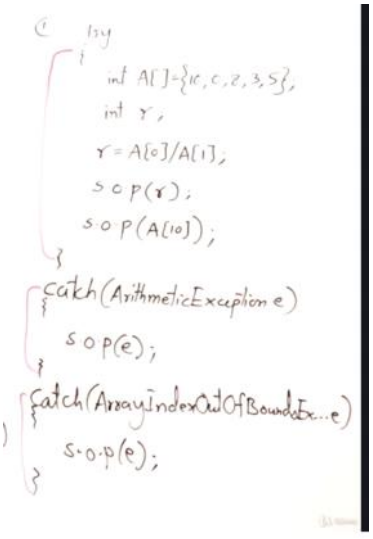




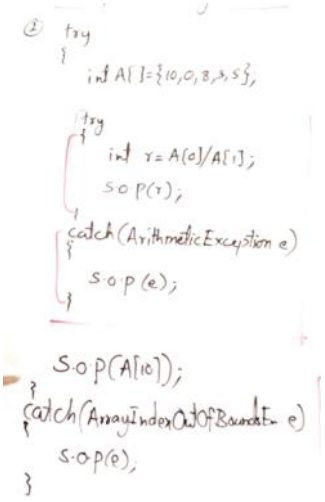
We want , even in case of exception, the program should run smoothly.



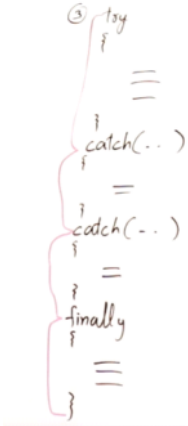
Multiple catch blocks



Here we have 2 exceptions, so we got 2 catch blocks.

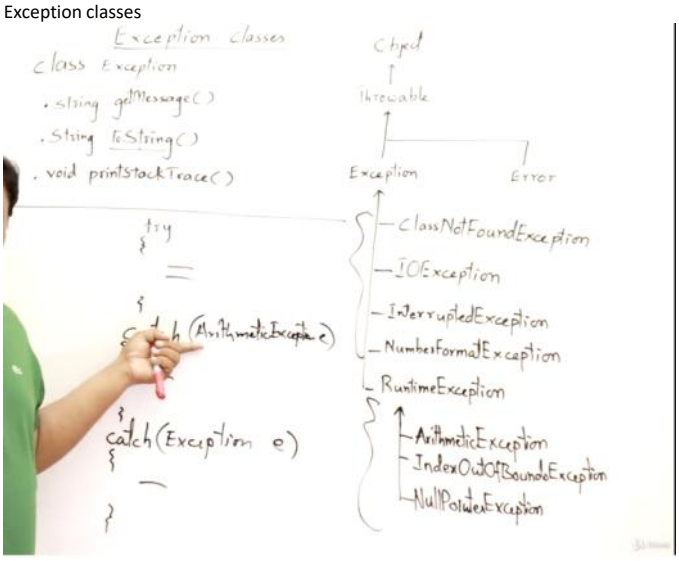


Same thing as before, but with multiple try and catch.



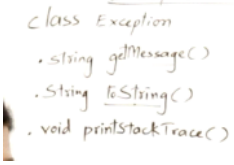
Finally block definitely gets executed.

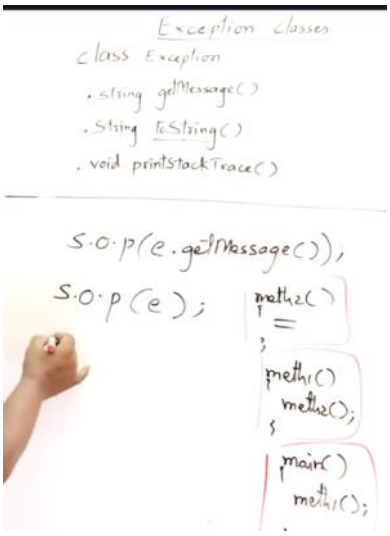
Exception classes



Here we can see Exception class is parent of all exceptions.
So in the first catch block, we can only catch Arithmetic Exceptions, but in the second catch Block we can handle all exceptions.

Exception class important methods-

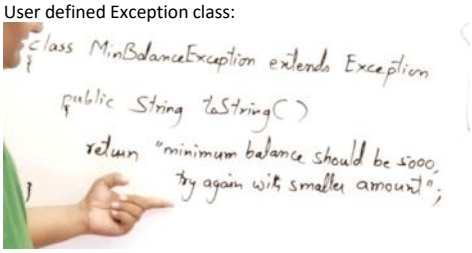




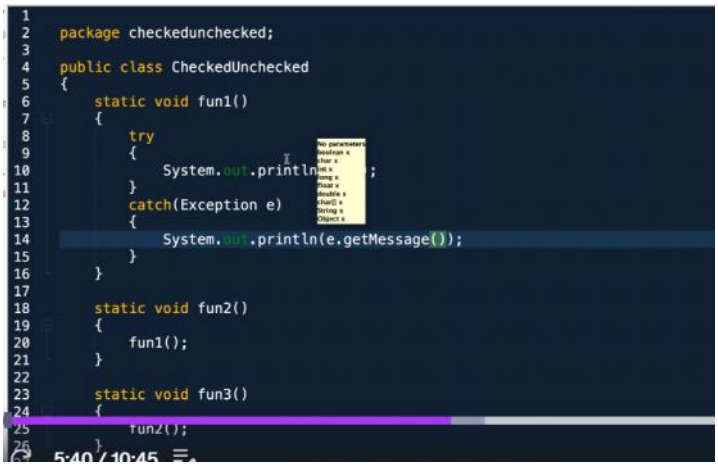
When u write e, toString() method is automatically called.
e.printStackTrace() will automatically print, u don't need to use SOP.

For writing a user-defined exception class, we have to write a class inheriting from Exception class.

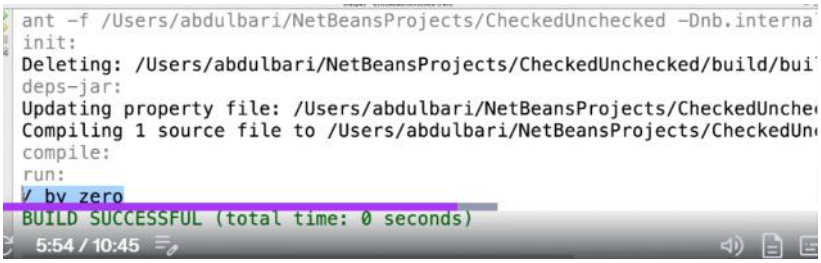
User defined Exception class:



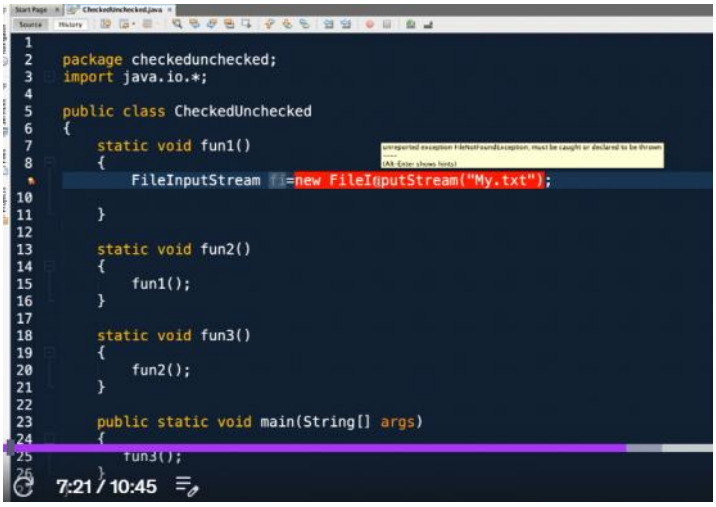
Checked and unchecked exceptions:



Compiling it, I got a message

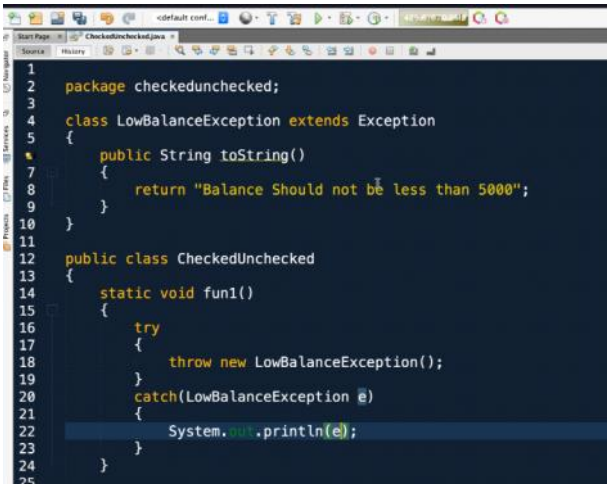


It is not giving any exception name.
If we just e, we get exception name too.



Here, file might not be there, so I must handle this exception.
So either write try/catch block or use throws.
These are called checked exceeptions.

Lets get into our defined exception



In that try-catch block, the code is **demonstrating how to raise (throw) and handle a custom checked exception**.
Let's break down what's happening:

1) What are we trying to do in the try block?

```
try {
   throw new LowBalanceException();
}
```

Inside try, we **intentionally create and throw** an instance of LowBalanceException.
☑ This simulates a situation like: *"Account balance went below the minimum allowed threshold (5000), so we signal an error."*
So the goal of the try block is:

- **to trigger an exception condition**
- **to transfer control to the corresponding catch block**

2) What are we doing in the catch block?

```
catch (LowBalanceException e) {
   System.out.println(e);
}
```

This catches the thrown LowBalanceException and prints it.

Why does System.out.println(e) print a message?

Because your exception class overrides toString():

```
public String toString() {
   return "Balance Should not be less than 5000";
}
```

So printing e calls e.toString() automatically, and the output becomes:

☑ **Balance Should not be less than 5000**

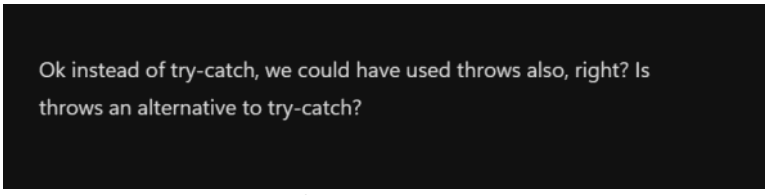
3) Overall purpose of the try-catch here

In one line:

We are forcing a custom exception to occur and then handling it gracefully by printing a meaningful error message.

More practically:

In a real banking-style program, you'd throw this exception only when a condition fails, like:
if (balance < 5000) throw new LowBalanceException();
Then the catch could log it, show it to the user, or stop a withdrawal, etc.



Yes — you *can* use throws instead of a try-catch in many cases, but it's not exactly a 1:1 "alternative."
Think of them as **two different ways to deal with an exception**:

- **try-catch = handle it here**
- **throws = don't handle it here; pass responsibility to the caller**

☑ Is throws an alternative to try-catch?

In a sense, yes

Both satisfy the compiler for **checked exceptions** (like your LowBalanceException extends Exception).
But they do **different jobs**:

try-catch (handle locally)

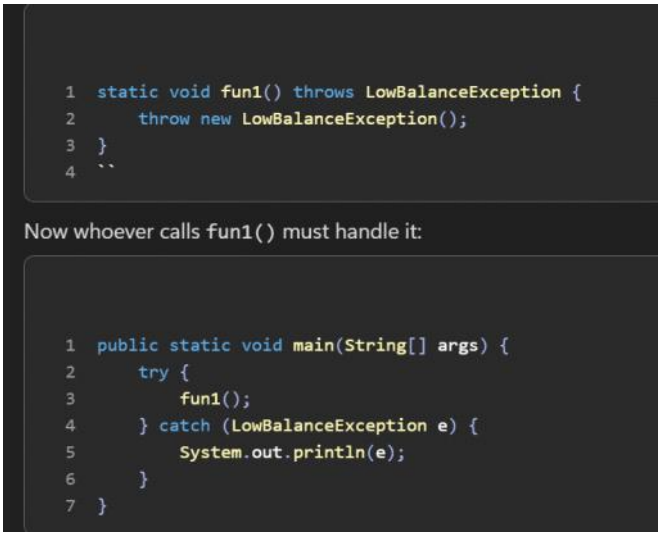
Use it when you want to:

- recover from the error
- show a user-friendly message
- log it
- continue execution safely

throws (propagate / delegate handling)

Use it when:

- this method can't/shouldn't decide what to do
- you want the caller (higher-level code) to decide
- you're building reusable library methods



```
static void fun1() throws LowBalanceException {
    throw new LowBalanceException();
}
``
```

Now whoever calls `fun1()` must handle it:

```
public static void main(String[] args) {
    try {
        fun1();
    } catch (LowBalanceException e) {
        System.out.println(e);
    }
}
```

}
Show more lines
☑ Here the handling happens in main() instead of inside fun1().

🔥 Key difference: "Handling" vs "Declaring"

throws does not handle the exception

It only *declares*:

"This method might throw this exception; caller must deal with it."

So if nobody catches it up the call stack, the program terminates with a stack trace.

why is Arithmetic exception an unchecked exception?
Great question — and this is one of the most important concepts in Java exception handling.\ Let's break it down clearly and simply.

☑ Why is ArithmeticException an *unchecked* exception?

Because **it represents programmer errors that can't be reasonably predicted or recovered from at compile time**, so Java leaves it to the **runtime** to detect.
ArithmeticException extends RuntimeException, and **all exceptions that extend RuntimeException are unchecked**.

🔍 1. Unchecked exceptions = programmer mistakes

Unchecked exceptions typically indicate errors such as:

- Dividing by zero
- Null pointer access
- Array index mistakes
- Bad cast
- Logic errors

These are **bugs in code**, not external conditions.

Example:

```
int a = 10 / 0; // ArithmeticException at runtime
``
```

Here the compiler cannot guarantee that the denominator is zero or non-zero — it depends on runtime values.
Java designers (James Gosling & team) wanted **unchecked exceptions** to represent these coding mistakes so developers don't have to handle them compulsorily.

🔍 2. Compile-time checking does NOT help here

Even if Java forced you to write:

```
try {
    int x = 10 / y;
} catch (ArithmeticException e) {}
```

You still wouldn't know if y becomes zero at runtime.
So compile-time enforcement would not prevent the issue — it would only make code ugly and pointless.
➡ **Therefore, Java lets it occur at runtime**, and if the program hasn't guarded against it, it fails.

🔍 3. Checked exceptions = recoverable situations

Checked exceptions represent conditions where:

- The problem is *external*
- The programmer *can* do something about it

Examples:

- File not found
- Network connection lost
- Database unavailable

These are **not programmer bugs** — they're environmental issues — so Java forces handling them.

🔍 4. Unchecked = performance

The designers of Java intentionally kept exceptions like ArithmeticException unchecked because:

- Forcing try-catch everywhere would have made Java **slow and extremely verbose**.
- Arithmetic operations happen constantly; wrapping each in try-catch would be ridiculous.

📌 Final Summary

ArithmeticException is unchecked because:

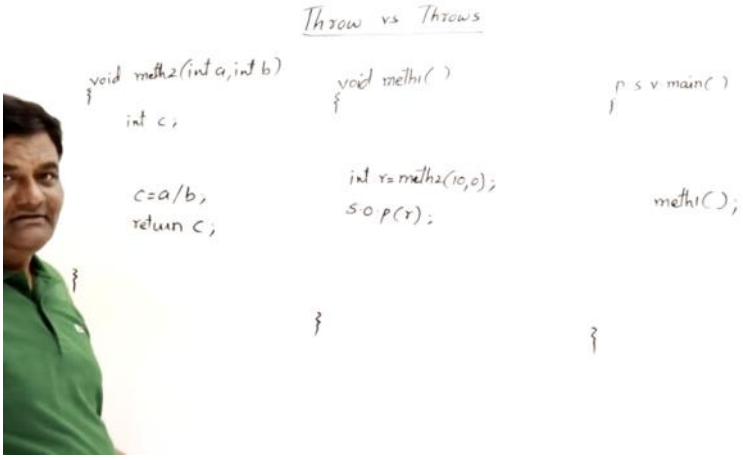
✓ It represents a programming error, not an external condition.
✓ It cannot be predicted or prevented reliably at compile time.
✓ Compiler enforcement would not help.
✓ Making it checked would clutter Java programs with useless try-catch blocks.
✓ Java expects developers to fix the logic, not handle the exception.

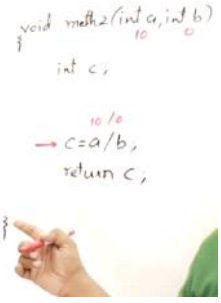
If you want, I can also explain:

- Why NullPointerException is also unchecked
- Why custom business exceptions often *should* be checked
- When you should choose checked vs unchecked for your *own* exceptions

Just tell me!

Throw vs Throws

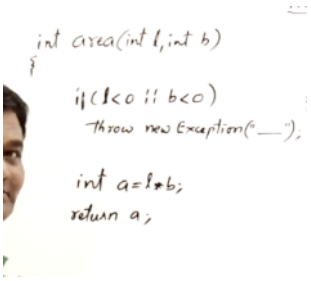




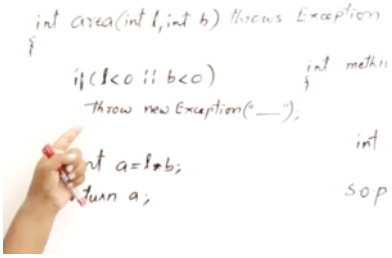
In this red line, exception will be raised- it will look for try-catch block but it ain't there. So it will go to the calling method meth1. so then we gotta write try-catch in meth1. but we see there is no try-catch in meth1 as well. So the exception will pass on to the method which called meth1() - so now exception is passed onto the main method as about the above example. Then program will crash as no try-catch. This can be illustrated by printstacktrace()

Now lets look at the next example:

Now if negative length or breadth is passed about in the main() -> The area can't be calculated - So we have to tell the calling method around this. SO gotta throw exception. What do we throw? -> object of Exception.

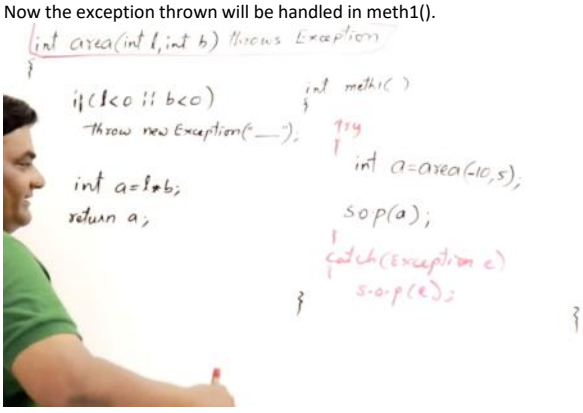


The purpose of throwing is to throw it to the calling method.
What happens when I throw exception?

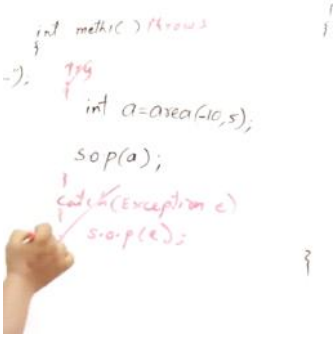


We defined throws in the method signature because this method throws exception.

The method that throws checked exception must use 'throws' in its signture.
Now the exception thrown will be handled in meth1().

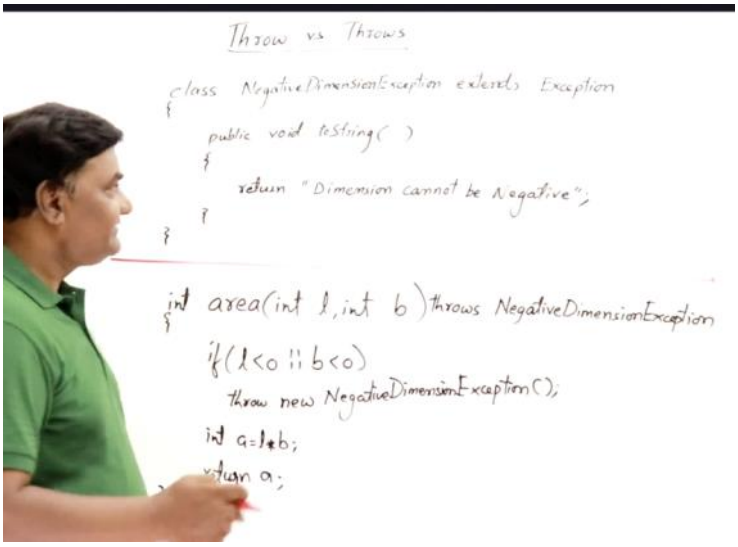


Ok now if I don't wanna catch the exceptions in meth1() also, rather in main(), then? Then write throws in meth1() also.

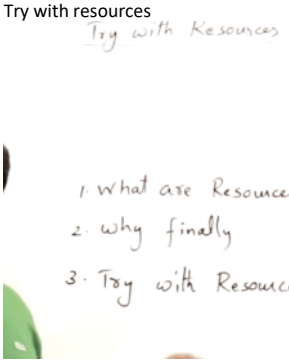


If a method doesn’t wanna handle exception, then easy way is throw exception.

Now we will define our own exception class.



Try with resources



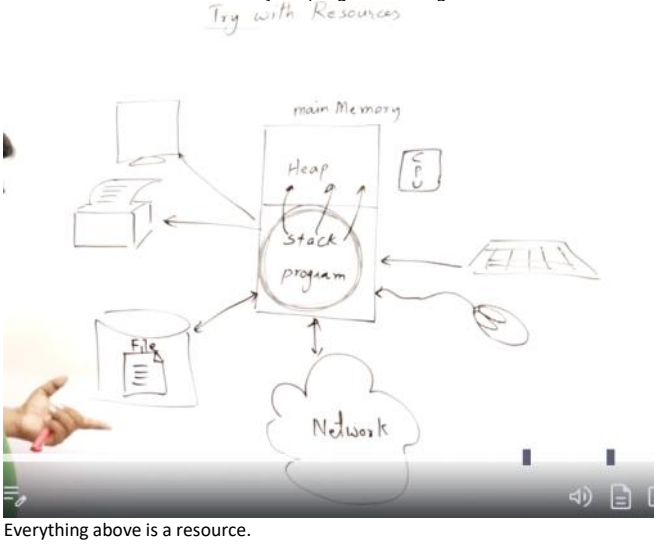
All the program that is outside the program is a resource. SO heap is a resource.

What is the heap? Why is it a resource?

The **heap** is not part of your program's code.
It is part of the **system memory** managed by the **operating system**.
Your program **requests** memory from the operating system:

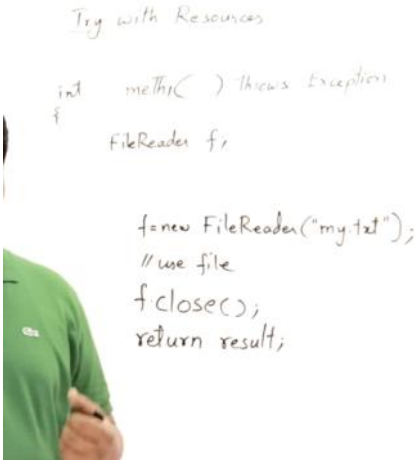
- In C → using malloc() or new
- In Java/Python → the runtime allocates memory for objects

Because the heap comes from the OS (outside the program),
it is considered a **resource** that your program is using.

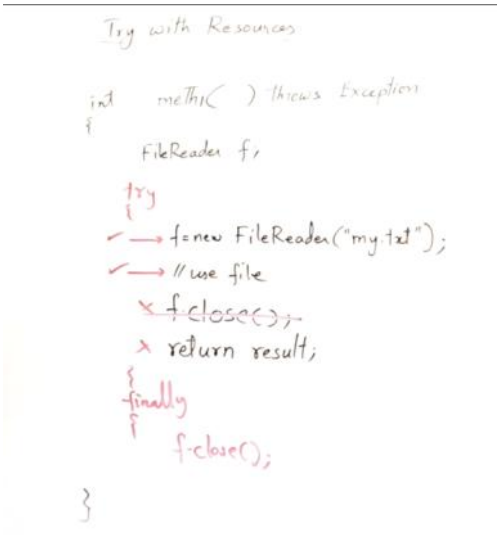


Everything above is a resource.
Db is also a resource.
When u need a resource, acquire it, when not in need, close it.

Lets see an use of 'finally' in here.



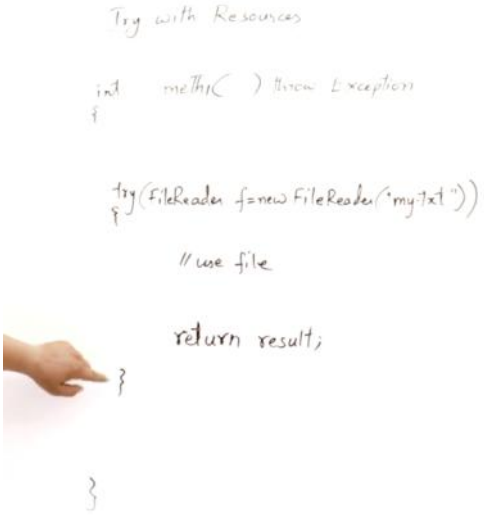
If exception arises before f.close(), then nobody closes the resource.
The program is still holding the file without releasing it.
We have to make sure cleanup is always done
So we have to use try-finally. No need for catch here.



In case of no-exception->Before return result; finally block will work.
In case of exception-> Finally block will execute and then throw the exception.

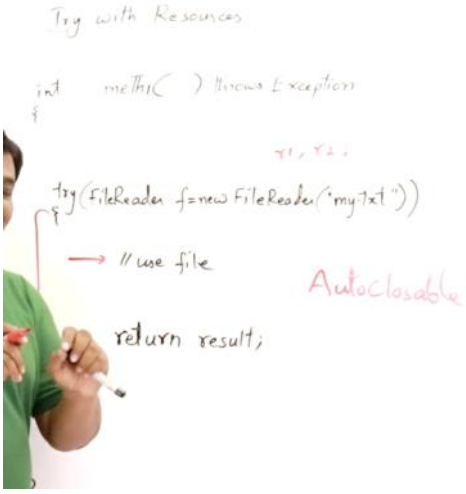
Thus resources used will be closed and that will be ensured.

Now we will use try-with resources.

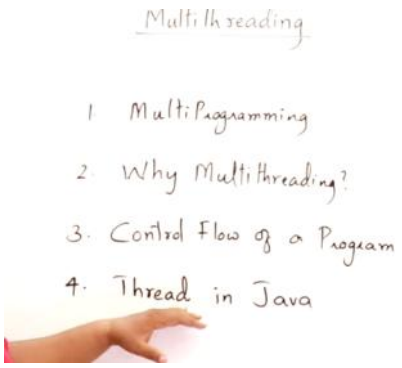


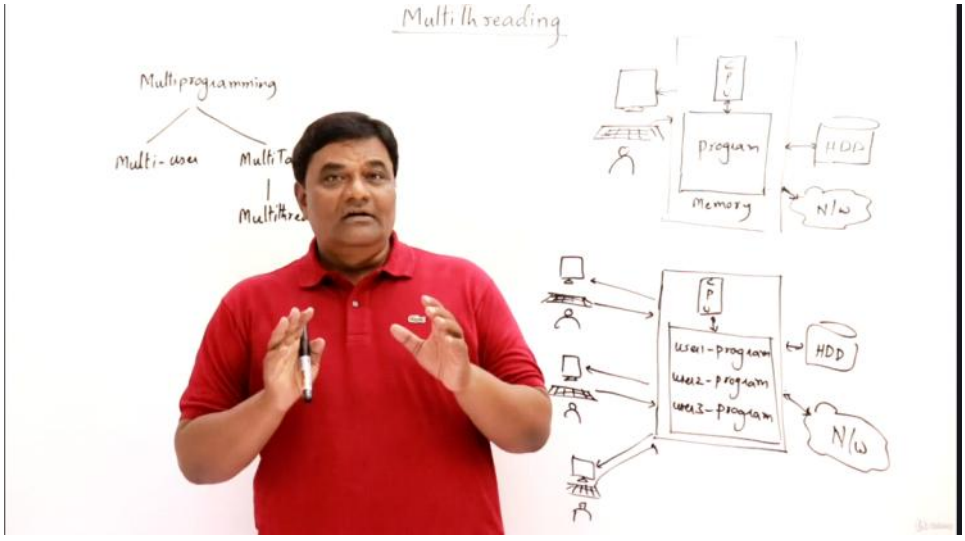
Here,we can create resources along with the try itself. And it will get automatically closed once the method is ending.
Resource will be automatically closed.

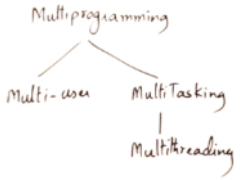
So we don't need a finally block anymore.



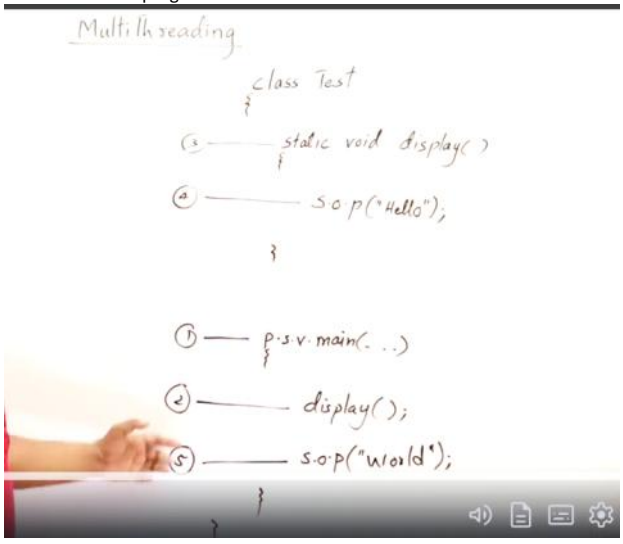
Multithreading





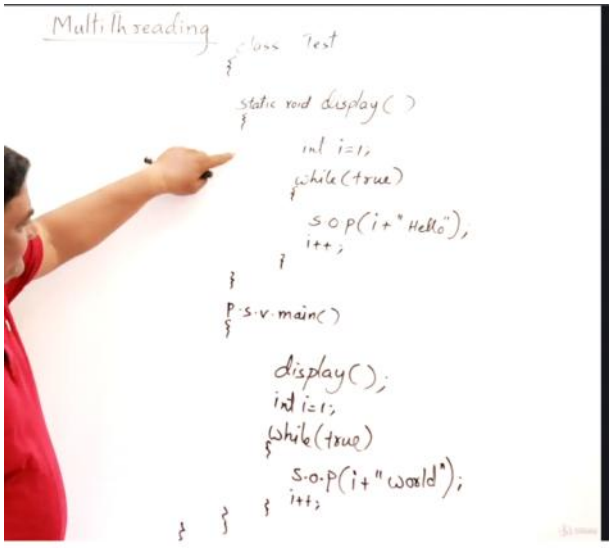


Control flow of a program



Here above is single flow of control

Now look at this code



The flow of control stays around the display method infinitely and the rest of the statements of the Main method are never executed.

But I want both the infinite loops to run simultaneously.

I want display() as a separate thread since main() is the entry point.

Hence, we need 2 flow of controls here.

So, few times Hello should appear, and few times world should appear.

Thus we have to run display() as a thread.

Multithreading in Java

1. Thread class
2. Runnable Interface

A class can only extend from one class.
So, either we use thread class.
Or, if we are extending some other class and still want to use thread class, then use Runnable Interface.
Because java can implement multiple interfaces.

Now we will be using Thread class.

```
class Test extends Thread
{
    public void run()
    {
        int i=1;
        while(true);
        {
            s.o.p(i+"Hello");
            i++;
        }
    }

    P.s.void main()
    {
        Test t=new Test();
        t.start();
        int i=1;
        while(true)
        {
            s.o.p(i+"world");
            i++;
        }
    }
}
```

```
class MyThread extends Thread
{
    public void run()
    {
        int i=1;
        while(true)
        {
            s.o.p(i+"Hello");
            i++;
        }
    }
}

class Test
{
    p.s.v.main()
    {
        MyThread t=new MyThread();
        t.start();
        int i=1;
        while(true)
        {
            S.o.p(i+"world");
```

Now we get all the features of the thread class from this block below:

```
class MyThread extends Thread
{
  public void run()
  {
    int i=1;
    while(true)
    {
      s.o.p(i+"Hello");
      i++;
    }
  }
}
```

We must override run() here. Whatever logic we want in a thread, we must write em in a run() method.
Just like main() is the starting point of a program, run() is the starting point of a thread.

```
class Test
{
  p.s.v.main()
  {
    MyThread t=new MyThread();
    t.start();
    int i=1;
    while(true)
    {
      S.o.p(i+"world");
```

Here t=new MyThread - > thread is created.
Now for running the thread, we should call start() method (not the run() method).

These 2 lines are starting the thread.

```
MyThread t=new MyThread();
t.start();
```

So, after thread starts, will run a separate control flow. The 'Hello' will be printing.
After thread starts, main() control flow will pass over to the next line and continues printing 'world'.

So now both loops will be running and both will keep on printing.

Now lets see the same thing, but instead of 2 classes, we would be using 1 class:

```
class Test extends Thread
{
  public void run()
  {
    int i=1;
    while(true);
    {
      s.o.p(i+"Hello");
      i++;
    }
  }

  P.s.void main()
  {
    Test t=new Test();
    t.start();
    int i=1;
    while(true)
    {
      s.o.p(i+"world");
      i++;
    }
  }
}
```

Multithreading Using Runnable Interface

```
class My implements Runnable
{
  public void run()
  {
    int i=1;
    while(true)
    {
      s.o.p(i+"Hello");
      i++;
    }
  }
}
```

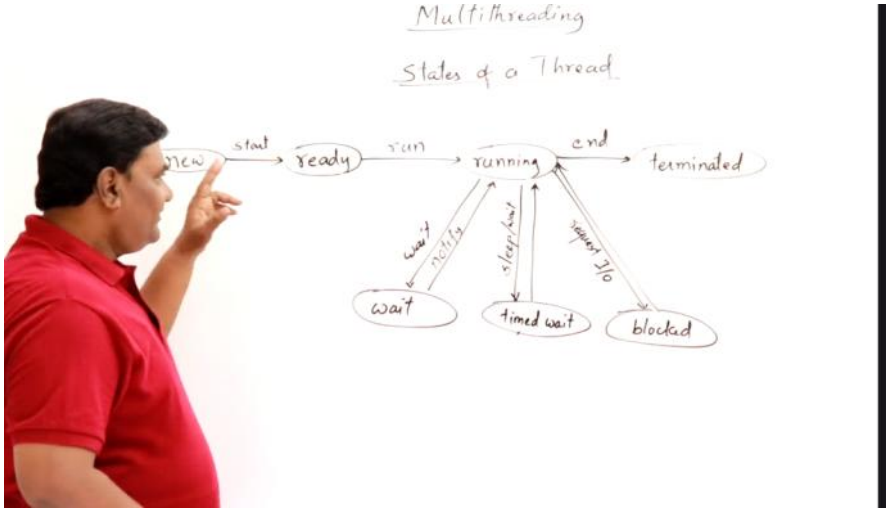
```
class My implements Runnable
{
  public void run()
  {
    int i=1;
    while(true)
    {
      s.o.p(i+"Hello");
      i++;
    }
  }
}
class Test
{
  p.s.v.main()
  {
    My m=new My();
    Thread t=new Thread(m);
    t.start();
    int i=1;
    while(true){s.o.p(i+"world"); i++;}
  }
}
```

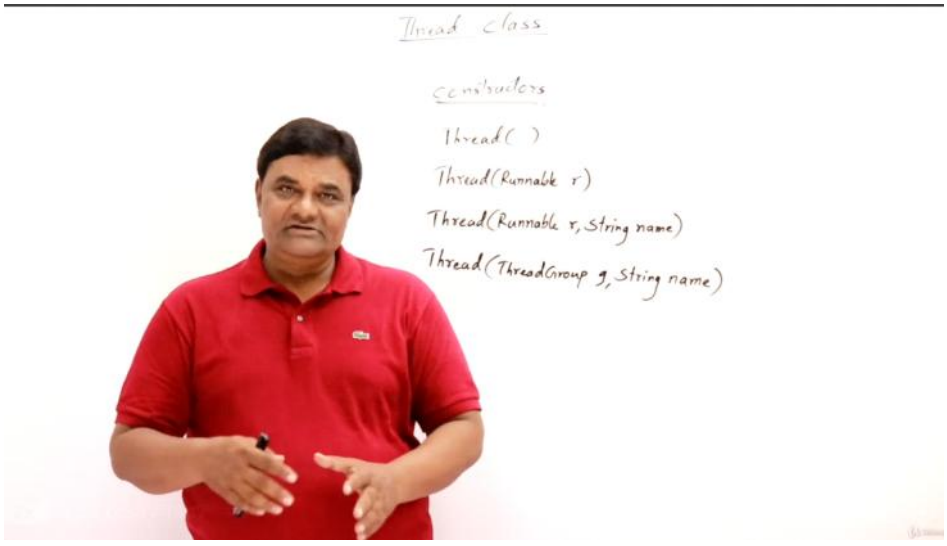
Runnable is capable of running threads - but it itself can't run it.
Here, My is not a thread, it cannot run by itself.
Then How to run it?
I need an object of thread.
Then attaching object if the class to that thread.
So, when the thread is running, the object will also run.
E.g - if thread is a horse, the object is a cart attached to that horse

Same can be achieved with a single class also like below:

```
class Test implements Runnable
{
  public void run()
  {
    int i=1;
    while(true)
    {
      s.o.p(i+"Hello");
      i++;
    }
  }
  p.s.v.main()
  {
    Test m=new Test();
    Thread t=new Thread(m);
    t.start();
    int i=1;
    while(true){s.o.p(i+"world"); i++;
  }
}
```

States of thread





Thread class

constructors

Thread( )

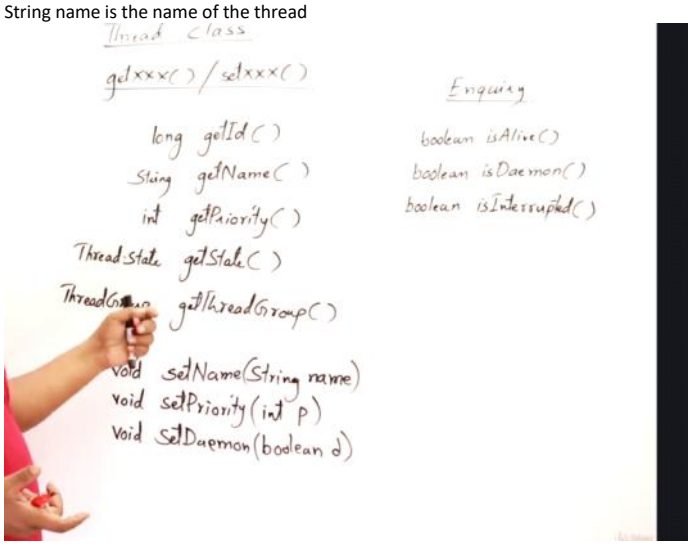
Thread(Runnable r)

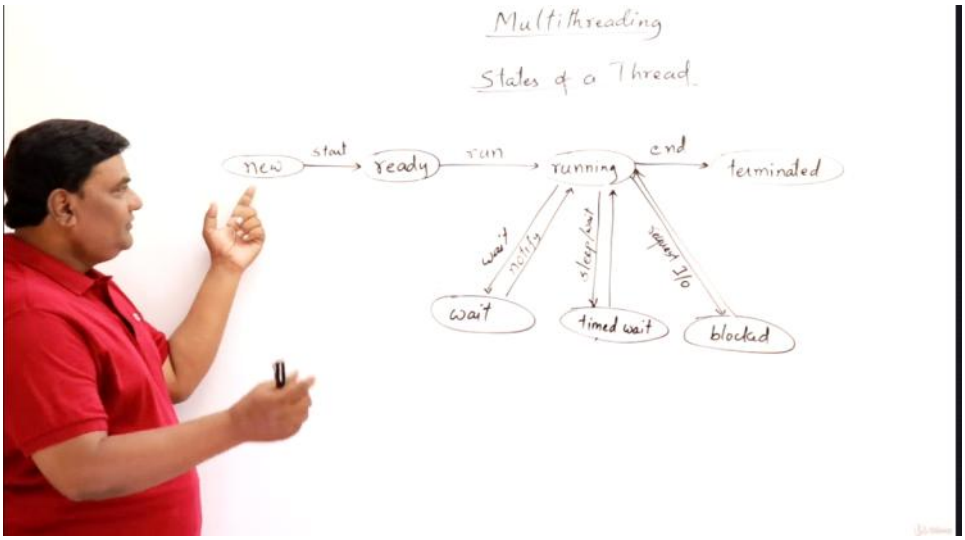
Thread(Runnable r, String name)

Thread(ThreadGroup g, String name)

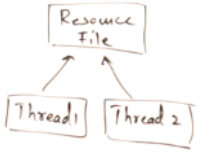
Thread(String name)

String name is the name of the thread



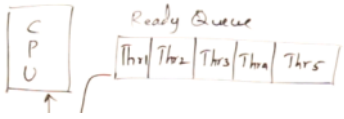


Wait and blocked states demo:

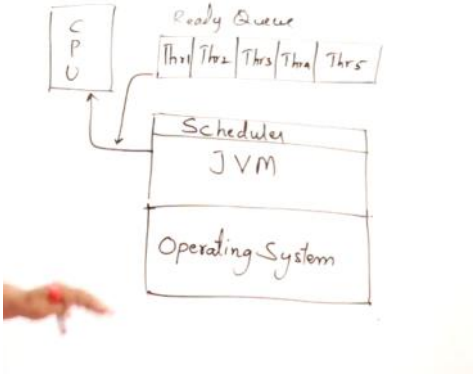


We can't allow both threads to access the same resource at the same time.
Resource should be handled by 1 thread at a time.

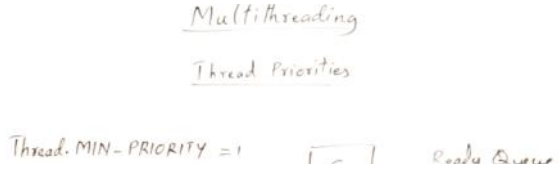
Thread Priorities
Suppose we have some threads in a ready queue.

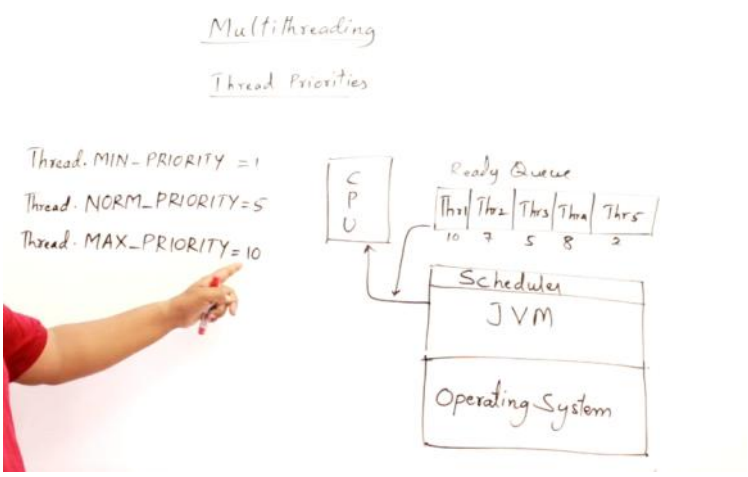


JVM will maintain this ready queue. The scheduler in JVM will maintain that.

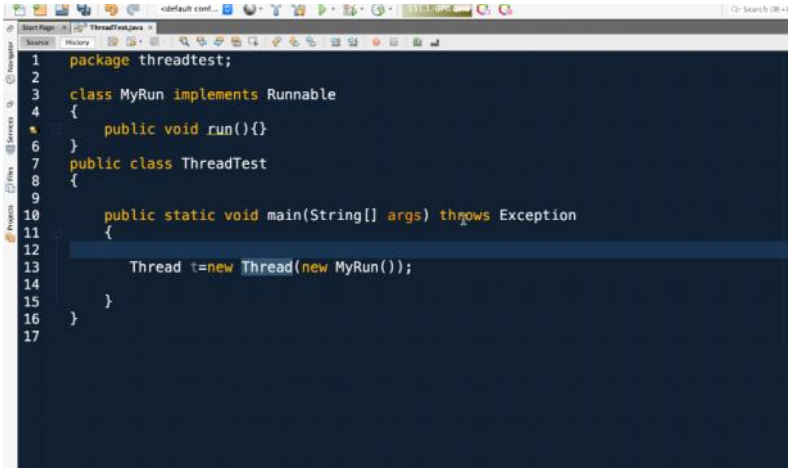


At a time CPU will run 1 thread at a time.
Suppose every threads get 1 seconds of time by CPU
Scheduler manages that
Now java supports priority from 1 to 10.
So, now the thread having higher priority should get more time.
If u don't set priority of a thread it wil have prority 5 by default
E.g. when u type on docs, the keys pressed are displayed -> this thread has higher priority,
The autosave which will be done by thread would have lower priority.
The multithreading environment is given by the OS, but java has its own multithreading- JVM
Can take care of it.

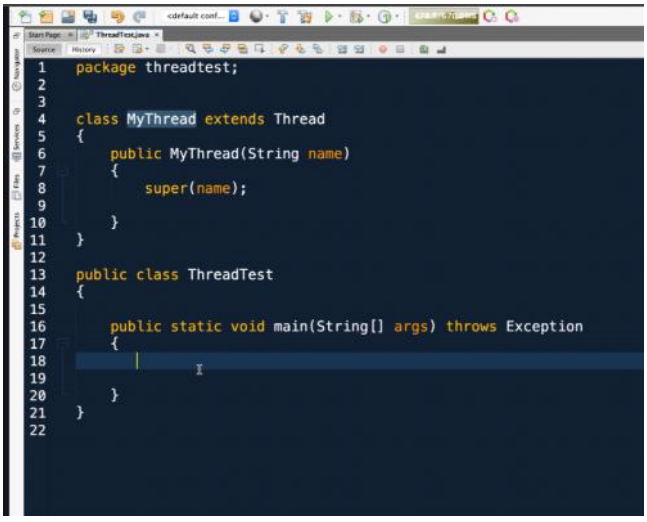




Thread Methods



```
1  package threadtest;
2
3  class MyRun implements Runnable
4  {
5      public void run(){}
6  }
7  public class ThreadTest
8  {
9
10     public static void main(String[] args) throws Exception
11     {
12
13        Thread t=new Thread(new MyRun());
14
15     }
16 }
17
```



```
1  package threadtest;
2
3
4  class MyThread extends Thread
5  {
6      public MyThread(String name)
7      {
8          super(name);
9
10     }
11 }
12
13 public class ThreadTest
14 {
15
16     public static void main(String[] args) throws Exception
17     {
18
19
20     }
21 }
22
```

```java
package threadtest;

class MyThread extends Thread
{
    public MyThread(String name)
    {
        super(name);

    }
}

public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {
       MyThread t=new MyThread("My Thread 1");

    }
}
```

Above is all about constructors

Now check some methods.

```java
package threadtest;

class MyThread extends Thread
{
    public MyThread(String name)
    {
        super(name);

    }
}

public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {
       MyThread t=new MyThread("My Thread 1");

       System.out.println("ID "+t.getId());
       System.out.println("Name "+t.getName());
       System.out.println("Priority "+t.getPriority());
       System.out.println("State "+t.getState());
       System.out.println("Alive "+t.isAlive());

    }
}
```

Output:

```
Updating property file: /Users/abdulbari/N
Compiling 1 source file to /Users/abdulbar
compile:
run:
ID 12
Name My Thread 1
Priority 5
State NEW
Alive false
BUILD SUCCESSFUL (total time: 0 seconds)
```

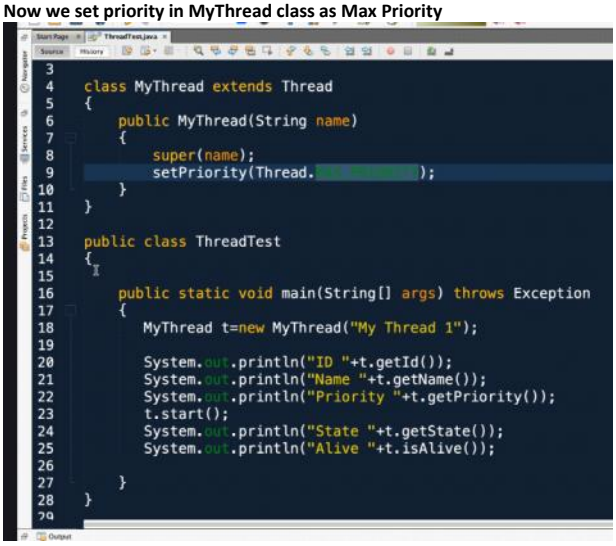
Now lets make some change, before getState(), we will start running the thread and See what's printed.

```java
public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {
       MyThread t=new MyThread("My Thread 1");

       System.out.println("ID "+t.getId());
       System.out.println("Name "+t.getName());
       System.out.println("Priority "+t.getPriority());
       t.start();
       System.out.println("State "+t.getState());
       System.out.println("Alive "+t.isAlive());

    }
}
```

```
Updating property file: /Users/abdulbari/
Compiling 1 source file to /Users/abdulba
compile:
run:
ID 12
Name My Thread 1
Priority 5
State RUNNABLE
Alive false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Now, its changed to runnable.

Now we set priority in MyThread class as Max Priority



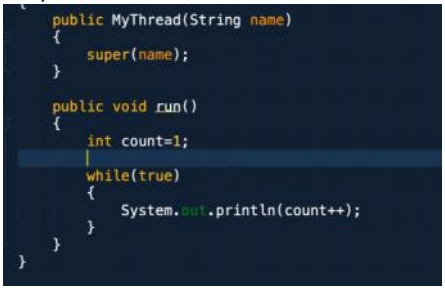
```
3
4   class MyThread extends Thread
5   {
6       public MyThread(String name)
7       {
8           super(name);
9           setPriority(Thread.MAX_PRIORITY);
10      }
11  }
12
13  public class ThreadTest
14  {
15
16      public static void main(String[] args) throws Exception
17      {
18          MyThread t=new MyThread("My Thread 1");
19
20          System.out.println("ID "+t.getId());
21          System.out.println("Name "+t.getName());
22          System.out.println("Priority "+t.getPriority());
23          t.start();
24          System.out.println("State "+t.getState());
25          System.out.println("Alive "+t.isAlive());
26
27      }
28  }
```

Output:

```
Updating property file: /Users/abdulbari/Ne
Compiling 1 source file to /Users/abdulbari
compile:
run:
ID 12
Name My Thread 1
Priority 10
State RUNNABLE
Alive false
BUILD SUCCESSFUL (total time: 0 seconds)
```

We are getting Priority 10.

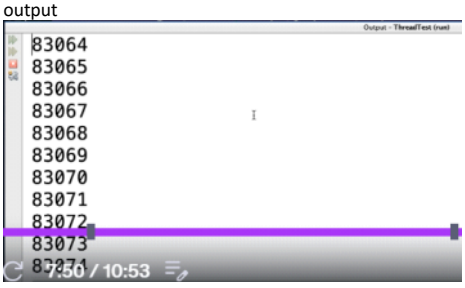
Now,



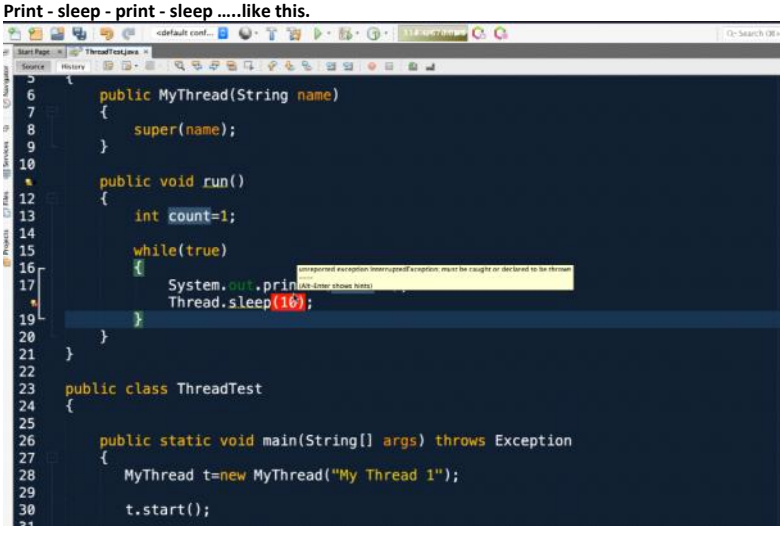
```
    public MyThread(String name)
    {
        super(name);
    }

    public void run()
    {
        int count=1;

        while(true)
        {
            System.out.println(count++);
        }
    }
}
```

```
    public MyThread(String name)
    {
        super(name);
    }

    public void run()
    {
        int count=1;

        while(true)
        {
            System.out.println(count++);
        }
    }
}

public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {
       MyThread t=new MyThread("My Thread 1");

       t.start();

    }
```

output



We can see the infinite loop output printings is so fast. Suppose I wanna make it slow.
Print - sleep - print - sleep .....like this.

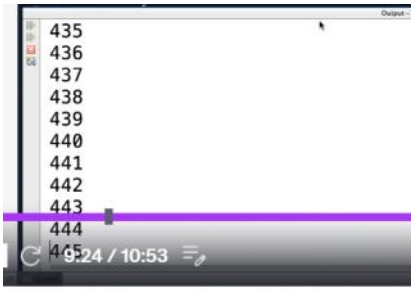


```
    public MyThread(String name)
    {
        super(name);
    }

    public void run()
    {
        int count=1;

        while(true)
        {
            System.out.prin
            Thread.sleep(16);
        }
    }
}

public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {
       MyThread t=new MyThread("My Thread 1");

       t.start();
```

But sleep() throws InterruptedException, so I gotta use try-catch.

```
    public MyThread(String name)
    {
        super(name);
    }

    public void run()
    {
        int count=1;

        while(true)
        {
            System.out.println(count++);
            try
            {
                Thread.sleep(10);
            }
            catch(InterruptedException e)
            {
                System.out.println(e);
            }
        }
    }
}
```

Now printing will be bit slower



Now we will do an interrupt in the main()

```
public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {
       MyThread t=new MyThread("My Thread 1");

       t.start();
       t.interrupt();
    }
}
```

o/p:

```
Compiling 1 source file to /Users/abdulbari/NetBeansPr
compile:
run:
1
java.lang.InterruptedException: sleep interrupted
2
3
4
5
6
```

But it was handled and printed after 1 is printed.

Thread:Daemon, Join and Yield

```
1   package threadtest;
2
4   class MyThread extends Thread
5   {
6      public void run()
7      {
8          int count=1;
9
10         while(true)
11         {
12             System.out.println(count++);
13
14         }
15     }
16  }
17
18  public class ThreadTest
19  {
20
21      public static void main(String[] args) throws Exception
22      {
23         MyThread t=new MyThread();
24         t.start();
25      }
26  }
```

Here, we can see main() is waiting for MyThread thread to finish.
Now we setDaemon before running the thread

```java
package threadtest;

class MyThread extends Thread
{
    public void run()
    {
        int count=1;

        while(true)
        {
            System.out.println(count++);

        }
    }
}

public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {
       MyThread t=new MyThread();
       t.setDaemon(true);
       t.start();
    }
}
```

If we wanted to set Daemon in MyThread class, then we had to define constructor.

Now if we run the program, nothing is printed.
But as the main() ends, the thread also terminates

Now, lets roll program around like this: Before main() terminates, it will sleep for a while.

```java
public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {
       MyThread t=new MyThread();
       t.setDaemon(true);
       t.start();

       try{Thread.sleep(100);}catch(Exception e){}
    }
}
```

o/p:

```
16036
16037
16038
16039
16040
16041
16042
16043
16044
BUILD SUCCESSFUL (total tim
3:32 / 10:21
```

Now we can see some printings.

Now we shall see join.
Main() will wait for other threads to complete.
We need reference of the main thread.

```java
Thread mainThread=Thread.currentThread();
```

Now we implement join.

```java
public class ThreadTest
{

    public static void main(String[] args) throws Exception
    {
       MyThread t=new MyThread();
       t.setDaemon(true);
       t.start();
       Thread mainThread=Thread.currentThread();
       mainThread.join();
    }
}
```

This will join any threads that's present- we not need to mention explicitly. So
Despite t is now a daemon thread, main() is waiting for threads.
o/p:

```
15    }
63577
63578
```

```
63577
63578
63579
63580
63581
63582
63583
63584
63585
63586
```

5:26 / 10:21

Now we will see yield.

```
class MyThread extends Thread

    public void run()
    {
        int count=1;

        while(true)
        {
            System.out.println(count++ +"My thread");

        }
    }


ublic class ThreadTest


    public static void main(String[] args) throws Exception
    {
       MyThread t=new MyThread();
       t.start();

       int count=1;

        while(true)
        {
            System.out.println(count++ +"Main");
```

o/p

```
36708My thread
36709My thread
36710My thread
39171Main
39172Main
36711My thread
36712My thread
36713My thread
36714My thread
39173Main
39174Main
```

Sometimes main, sometimes thread.
Now if we write yield() in any method, it will wait for other thread to execute.
Suppose we write yield() in main().

```
ublic class ThreadTest

    public static void main(String[] args) throws Exception
    {
       MyThread t=new MyThread();
       t.start();

       int count=1;

        while(true)
        {
            System.out.println(count++ +" Main");
            Thread.yield();

        }

    }
```

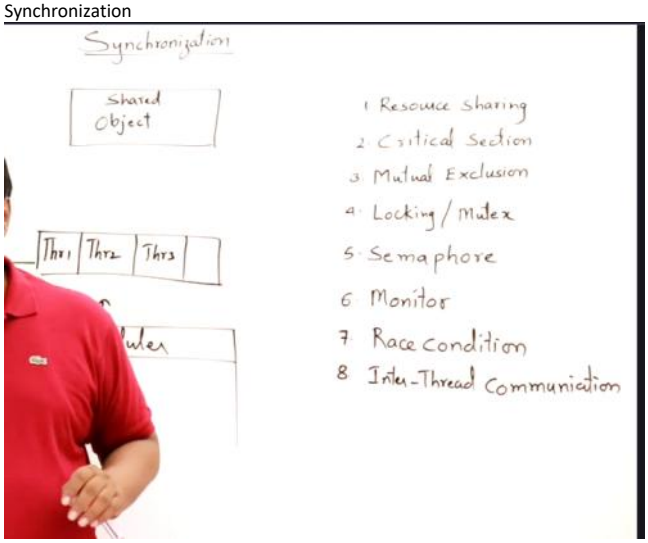
o/p:

```
21141 My thread
21142 My thread
13105 Main
21143 My thread
21144 My thread
21145 My thread
13106 Main
13107 Main
13108 Main
21146 My thread
```

```
21141 My thread
21142 My thread
13105 Main
21143 My thread
21144 My thread
21145 My thread
13106 Main
13107 Main
13108 Main
21146 My thread
21147 My thread
```

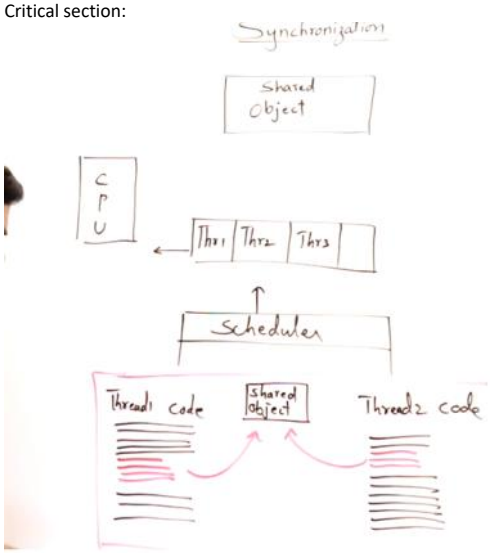
We can see My Thread is printed more no. of times.

Synchronization



Here we will see synchronization between threads.
Suppose more than one threads accessing the same file
A thread will have their own memory space stack
Heaps can be accessed by multiple threads -> shared resource.

Critical section:



Here the red lines are accessing the shared object.
These lines are called critical section.
That piece of code is the critical section.

Mutual Exclusion:
There would be problems around if both threads access the shared object simultaneously.
Mutual exclusion means that accessing of one threads prevents the accessing of another one.

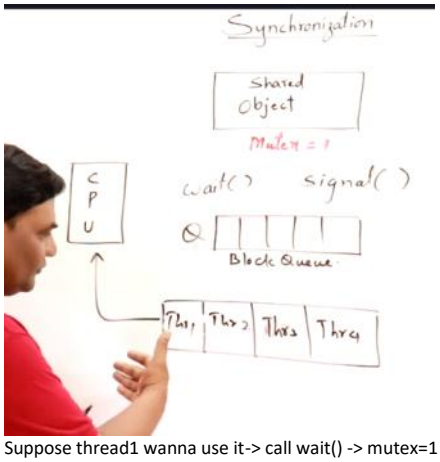
How to achieve mutual exclusion? There are 2 possibilities:

1. Somebody standing by the shared object will make sure that the shared object will be accessed by only one thread at a time. (like guard of ATM machine) -> locking/mutex, semaphore, monitor.
2. Threads will communicate with each other and take their turns. -> race-condition, inter-thread communication.

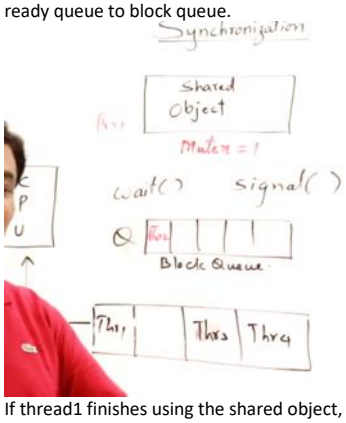
Locking/Mutex:

Mutex has its limitations. If 2 threads around and one wants to use it, mutex=1, thus locked.
But in a scenario where both saw a printer free (mutex=0), they came about and print it making mutex= 1 twice, which is a problem.

So somebody should watch there and take care of it -> here came semaphore.



Suppose thread1 wanna use it-> call wait() -> mutex=1
Now thread2 wanna use it -> call wait() -> wait() see mutex=1 i.e. somebody using it -> it goes from ready queue to block queue.

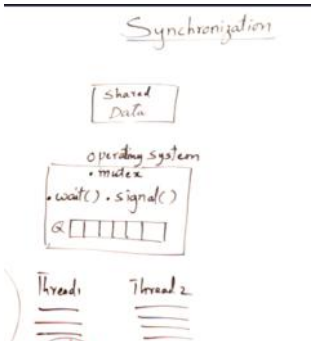


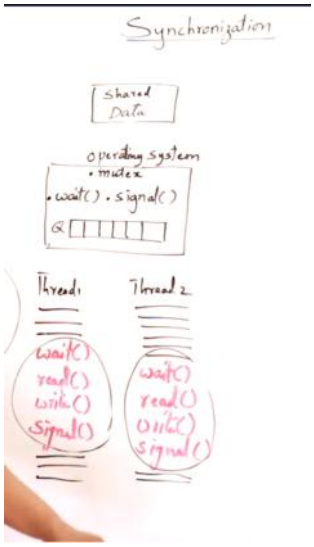
If thread1 finishes using the shared object, it will call signal(). Signal() will make mutex=0 i.e. free.
Thread2 will go to ready queue, and start using shared resource.

Semaphore is exactly the ATM security guard example.

OS is taking care of semaphore.

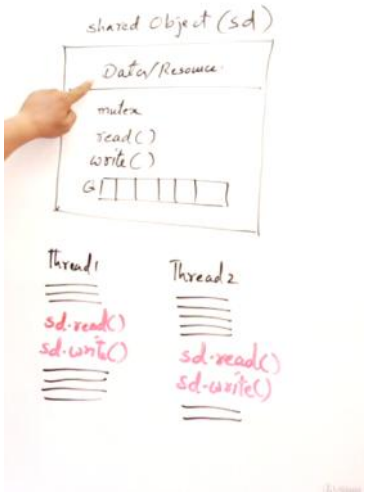
Monitor





Above is when semaphore involved.

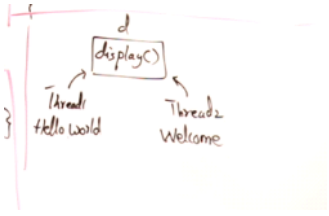
Now see this:



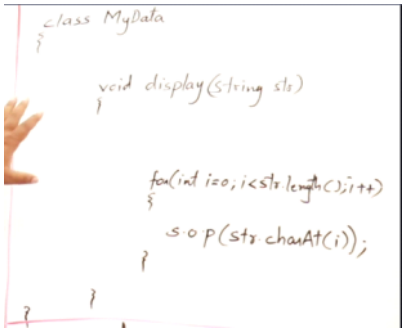
Here data is an object. Here locking and unlocking nis done by the object itself.
Threads are just calling the methods.
The object is taking the responsibility of read and write.
So complete mechanism is inside the object, unlike in previous example where
Operating System took care of the mechanisms.

This is Monitor.

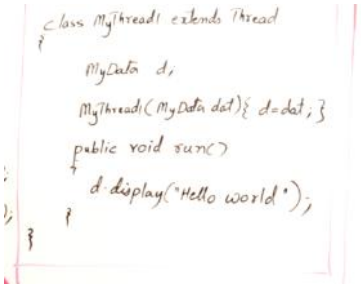
Multithreading Using Monitors



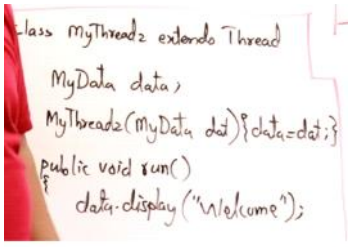
Here display() is the shared object.
Both threads will use single object - one print 'Hello World' and the other print 'Welcome'
If we don't use synchronization, the strings will be mixed.
The code (without synchronization):
Printing string by string:

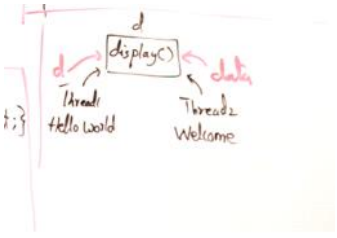


Thread1:

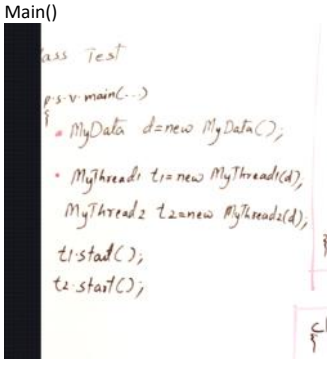


Thread 2:



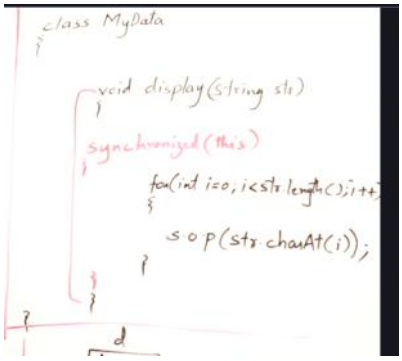


Main()



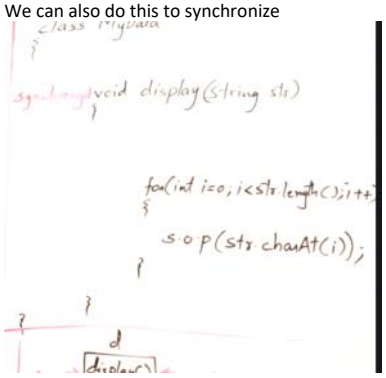
Now above code is not synchronised.

For synchronization:



Now letters of one thread will be printed at a time, instead of letters of both threads getting jumbled around one another.

We can also do this to synchronize



Got you — this code is basically a **multithreading + shared object** example, and the "confusing" part is **who calls what, and when**, especially because threads can run in any order.
I'll explain the **flow of control** like a story, step-by-step, and also point out the most common confusion points.

1) What each class does (in plain English)

☑ MyData

Has a method:

```
void display(String str) {
  for(int i=0; i<str.length(); i++) {
    System.out.print(str.charAt(i));
  }
}
```

Meaning:\ If you call display("Hello"), it prints characters one by one:
H e l l o (without spaces, just continuous output)
So it prints a string **character-by-character** in a loop.

☑ MyThread1 extends Thread

It stores a reference to a MyData object:

- Constructor gets MyData dat
- Saves it in d
- run() calls:

d.display("Hello world");
So **Thread1's job** is:\ 👉 print "Hello world" using the **shared** MyData object.

☑ MyThread2 extends Thread

Similar, but prints:
data.display("Welcome");
So **Thread2's job** is:\ 👉 print "Welcome" using the **same shared** MyData object.

☑ Test.main()

Creates **one MyData object** and **two threads**, both pointing to the same object, then starts them.

2) Now the REAL flow of control (important part)

Step A: Main thread starts executing main()

When the program begins, the **main thread** runs first.

```
MyData d = new MyData();
``
```

☑ A single object d is created.

Step B: Main thread creates thread objects

```
MyThread1 t1 = new MyThread1(d);
MyThread2 t2 = new MyThread2(d);
```

☑ Two Thread objects created\ ☑ Both store the SAME MyData d reference inside them

So memory-wise, it's like:

- t1.d → MyData object
- t2.data → same MyData object

Step C: start() is called (this is huge)

```
t1.start();
t2.start();
```

Key point: start() does NOT run run() directly in the same flow.

☑ start() tells JVM:

"Create a new OS-level thread, then call run() inside that new thread."

So after start():

- Thread1 is scheduled to run run()
- Thread2 is scheduled to run run()
- Main thread continues, and usually ends main()

3) What happens after both threads start?

The JVM thread scheduler decides order

This is the part you can't predict.

Possible orders:

Case 1: Thread1 runs fully first

Output becomes:

Hello worldWelcome

Case 2: Thread2 runs fully first

Output becomes:

WelcomeHello world

Case 3 (most common): They interleave (mix)

Because printing is character-by-character, the scheduler can switch between threads in the middle:

Example mixed output:

HeWlelcloom ew orld

Or:

WelHelco lo worldme

☑ This mixing happens because:

- Both threads call display()
- display() prints characters one at a time
- Thread scheduler can pause one thread and run the other anytime

4) The core "flow" in one diagram

Main thread flow

1. create MyData d
2. create t1(d)
3. create t2(d)
4. call t1.start() → launches Thread1
5. call t2.start() → launches Thread2
6. main ends

Thread1 flow (in parallel)

1. enters run()
2. calls d.display("Hello world")
3. loops printing each character

Thread2 flow (in parallel)

1. enters run()
2. calls d.display("Welcome")
3. loops printing each character

5) Why output becomes jumbled (the real reason)

Because **both threads share the same resource**: System.out (console output)\ and they print in small steps (one char at a time).

So interleaving is easy.

6) If you want “clean” output (no mixing)

Then you need to make display() **synchronized** so only one thread can execute it at a time.
Like:

```
synchronized void display(String str) {
  for(int i=0; i<str.length(); i++) {
    System.out.print(str.charAt(i));
  }
}
```

Now flow becomes:

- Thread1 enters display → locks MyData object
- Thread2 tries to enter display → waits
- Thread1 finishes → releases lock
- Thread2 runs → prints fully

So output will be **either**:
Hello worldWelcome
or
WelcomeHello world
…but never mixed.

7) Quick check: one more common mistake

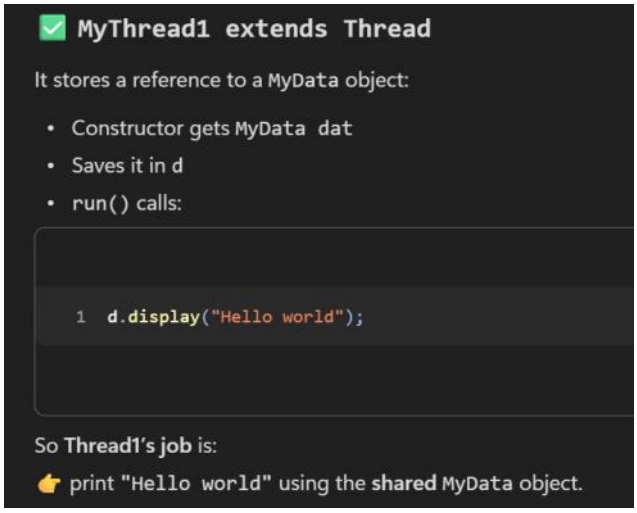
Sometimes people write public void sunc() by mistake instead of run().\ Only run() is called by start().
So ensure both thread classes have:

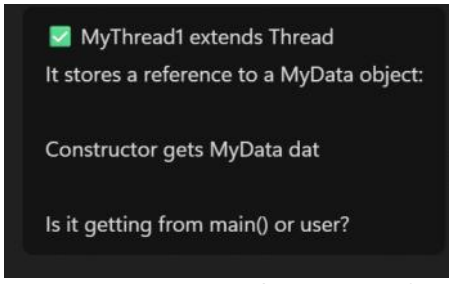
```
public void run() { … }
```

TL;DR (super short)

- main() creates one MyData object.
- Two threads share it.
- start() creates parallel threads that call run().
- Each run() calls display() which prints char-by-char.
- Scheduler can switch threads mid-loop → output gets mixed.
- synchronized fixes mixing by allowing one thread at a time.

If you want, paste the exact text of your code (or tell me if display() is synchronized in your notes),
and I’ll walk through a **sample execution timeline** like:
t1 prints H → switch → t2 prints W → switch → t1 prints e …
That usually makes the flow *click* instantly.





Great question — **it's coming from main()**, not from the user.

☑ Where does MyData dat come from?

Look at your main():

```
MyData d = new MyData();          // (1) main creates the MyData object
MyThread1 t1 = new MyThread1(d);  // (2) main passes it into Thread1 constructor
MyThread2 t2 = new MyThread2(d);  // (3) main passes the SAME object into Thread2
```

So when you write:
MyThread1(MyData dat) { d = dat; }

- dat receives whatever was passed during new MyThread1(d)
- That means: dat is literally the same object as d from main()

🤔 Think of it like "handing over a pointer/reference"

Java doesn't copy the object here. It passes a **reference** to the same object.
So this line:
d = dat;
just makes the thread's member variable point to that same MyData object created in main().

They pass the **same MyData object** to both threads to demonstrate what happens when multiple threads access the **same shared resource**.

🔁 Flow in one line

☑ **main creates MyData → main passes it to thread constructors → threads store the reference → later run() uses it**

❓ Is the user involved at all?

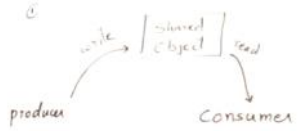
Not in this code.
There's **no input** like Scanner, no arguments from command line, nothing.\ The strings "Hello world" and "Welcome" are **hardcoded** inside run().

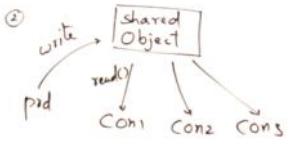
Quick mental model (super simple)

- main() = manager creates **one printer** (MyData)
- hires two workers (t1, t2)
- gives both workers the **same printer**
- both workers try printing → outputs can mix (because same printer)

If you want, I can also explain **why you pass the same MyData d** to both threads (shared resource), and how that leads to synchronization problems.

Inter- thread communication

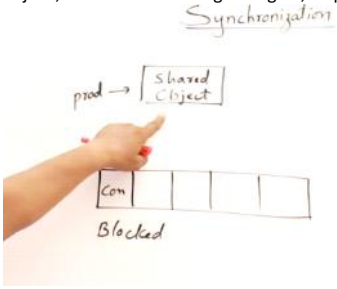


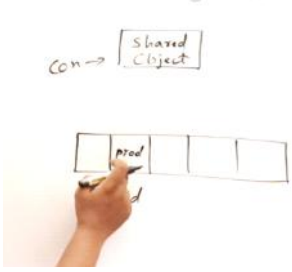


e.g. teacher write something, then u copy it down, then teacher again write something then u copy.
It should be synchronised. If teacher wrote something, u didn't copy and teacher rubbed it off and again wrote something, then there ain't be any synchronization.

Here, teacher is producer and student is consumer.

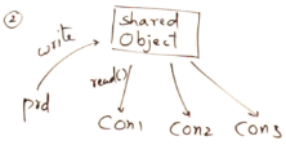
In java, we use variable flag. If flag =1, its producer's turn, if flag=0, its consumer's turn.





So, one time the producer is blocked, in other instance, consumer is blocked.
This happens through inter-thread communication.

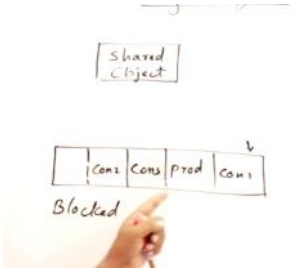
Now lets look at race condition



Here consumers should be consuming with round robin fashion, then producer produces it.
Here flag=0/1 won't be sufficient.
But here after suppose any consumer finishes task, it notifies, which might wake up any thread, because
In multithreading, we don't guarantee which thread runs first and which thread later.

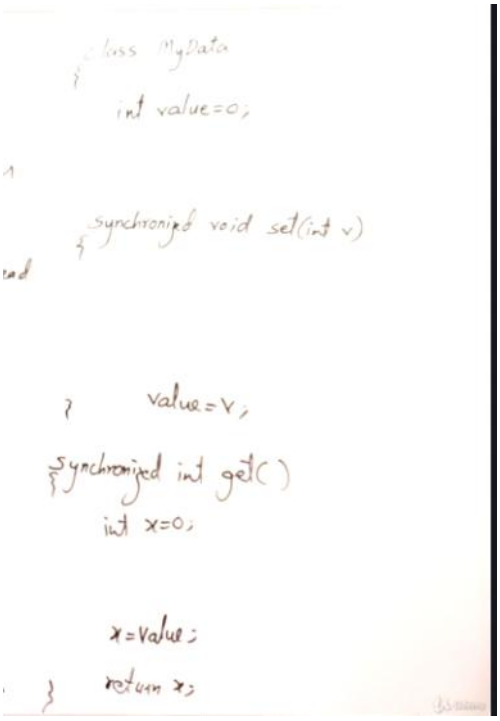
So, anybody may come up accessing shared resource -> this is race condition.

SO
Here we use count, count =1 -> consumer 1
Count-2 -> consumer 2
Count -3 -> consumer 3
Then again count goes 0 and producer accesses it.

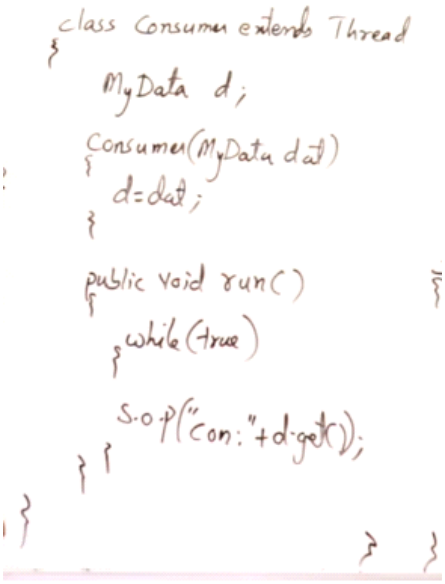


Example - inter - thread communication:

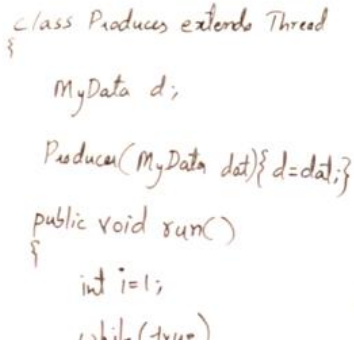
Shared class:



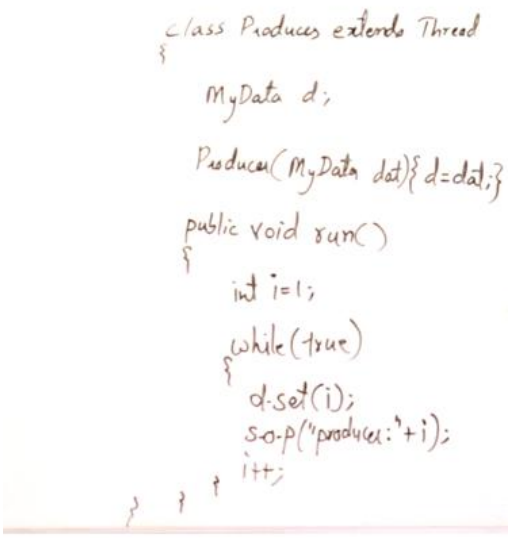
```
class MyData
{
    int value=0;

    synchronized void set(int v)
{
        value=v;
}
synchronized int get()
{
    int x=0;

    x=value;
    return x;
}
```



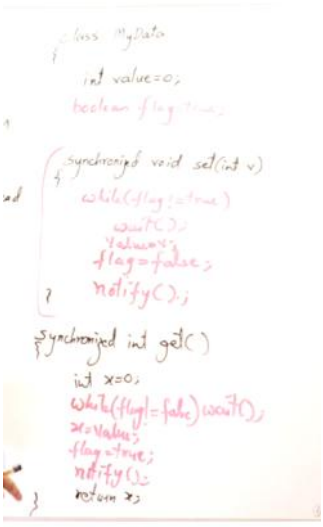
```
class Consumer extends Thread
{
    MyData d;
    Consumer(MyData dat)
    {
      d=dat;
    }
    public void run()
    {
      while(true)
      {
      S.o.p("Con:"+d.get());
    }
}
```



```
class Producer extends Thread
{
    MyData d;
    Producer(MyData dat){ d=dat;}
    public void run()
    {
        int i=1;
        while(true)
```



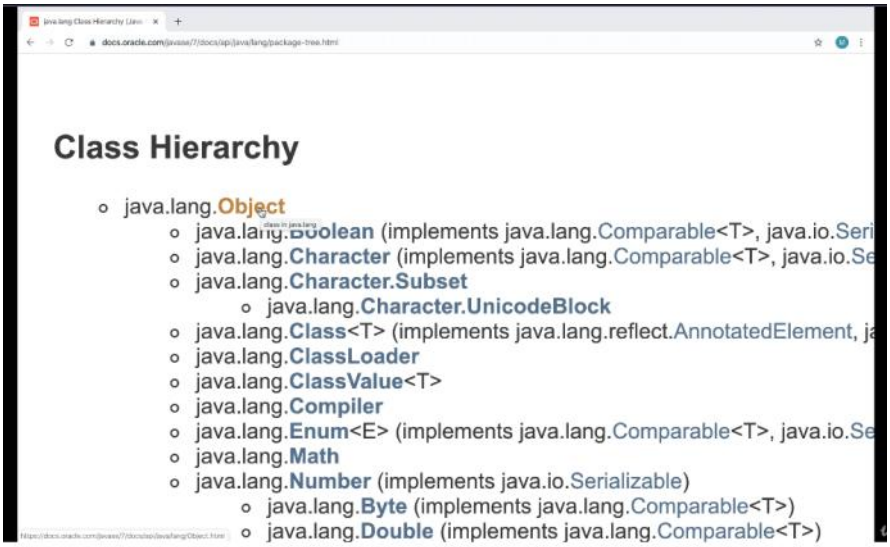
Here, the MyData class is not synchronised.
Lesss synchronize it.



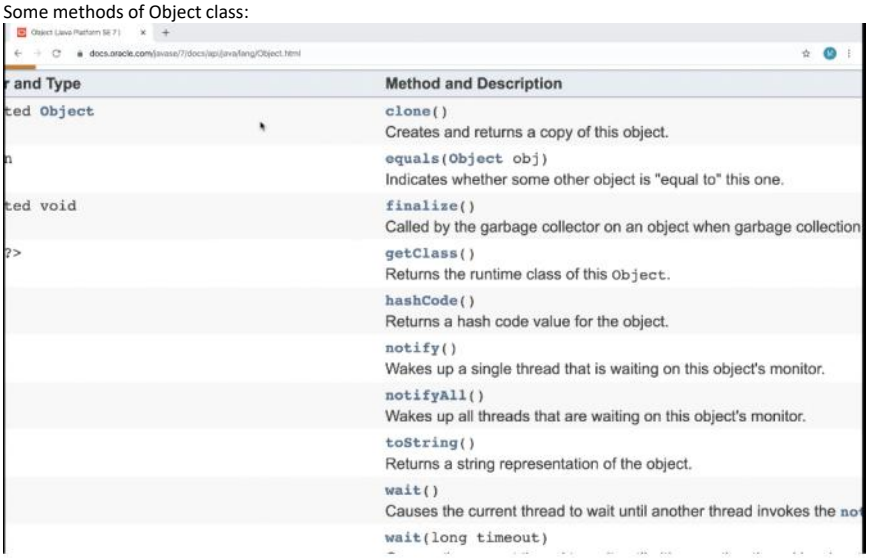
Java.lang package

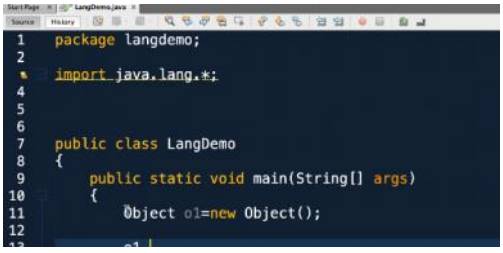
Object Class

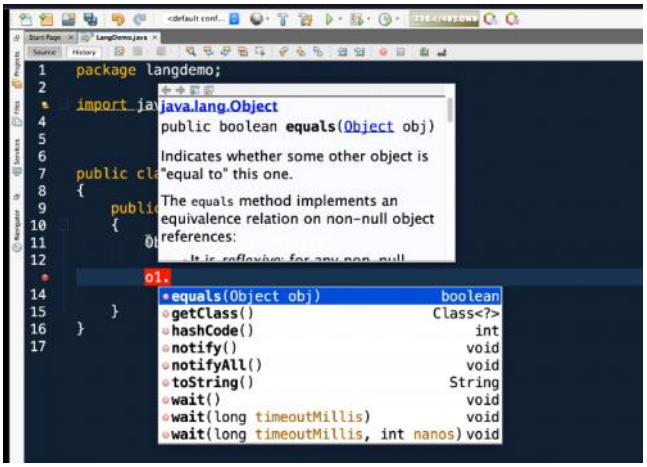
Any class in java is directly or indirectly inheriting from object class,. Not only classes from lang package, but every class is inheriting from Object class.



Some methods of Object class:







```
package langdemo;

import java.lang.*;


public class LangDemo
{
    public static void main(String[] args)
    {
        Object o1=new Object();

        System.out.println(o1);

    }
}
```

```
Compiling 1 source file to /Users/abdulbari/NetBeans
compile:
run:
java.lang.Object@4dc63996
BUILD SUCCESSFUL (total time: 0 seconds)
```

It is automatically calling o1.toString().

```
package langdemo;

import java.lang.*;


public class LangDemo
{
    public static void main(String[] args)
    {
        Object o1=new Object();

        System.out.println(o1.toString());

    }
}
```

```
Compiling 1 source file to /Users/abdulbari/NetBeansProjects
compile:
run:
java.lang.Object@4dc63996
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
package langdemo;

import java.lang.*;


public class LangDemo
{
    public static void main(String[] args)
    {
        Object o1=new Object();
        Object o2=new Object();

        System.out.println(o1.equals(o2));

    }
}
```

```
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/
compile:
run:
false
BUILD SUCCESSFUL (total time: 0 seconds)
```

False because these two are different objects.

```
package langdemo;

import java.lang.*;


public class LangDemo
{
    public static void main(String[] args)
    {
        Object o1=new Object();
```

```java
package langdemo;

import java.lang.*;


public class LangDemo
{
    public static void main(String[] args)
    {
        Object o1=new Object();
        Object o2=o1;

        System.out.println(o1.equals(o2));

    }
}
```

```
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/LangDemo
compile:
run:
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
package langdemo;

import java.lang.*;


class MyObject
{
    public String toString()
    {
        return "My Object";
    }

}

public class LangDemo
{
    public static void main(String[] args)
    {
        MyObject o1=new MyObject();

        System.out.println(o1);
    }
}
```

Overriden toString()

```
Compiling 1 source file to /Users/abdulbari/NetBe
compile:
run:
My Object
BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
    }

    public int hashCode()
    {
        return 100;
    }
}

public class LangDemo
{
    public static void main(String[] args)
    {
        MyObject o1=new MyObject();
        MyObject o2=new MyObject();

        System.out.println(o1.equals(o2))

    }
}
```

Here both objects have eqiual hashcode

```
}
Compiling 1 source file to /Users/abdulbari/Net
compile:
run:
false
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
    public boolean equals(Object o)
    {
        return this.hashCode()==o.hashCode();
    }
}
```

Overriden equals()

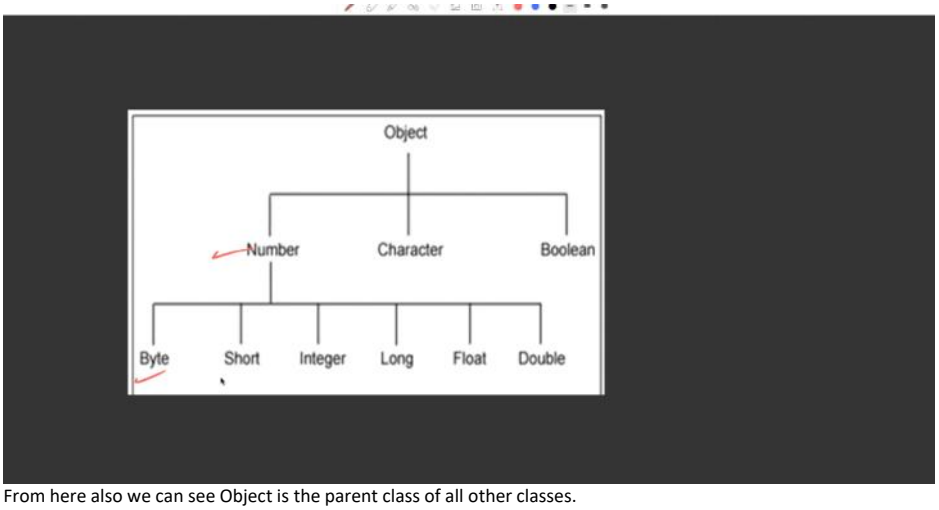
```
public class LangDemo
{
    public static void main(String[] args)
    {
        MyObject o1=new MyObject();
        MyObject o2=new MyObject();

        System.out.println(o1.equals(o2));
```

```
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/Lar
compile:
run:
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Wrapper class
Java is an object oriented language. But 'int' isn't object - it is primitives.
Because if we define - int i ;
If we type i dot operator something, we should have got some methods.

```
1
2   package wrapperdemo;
3
4   public class WrapperDemo
5   {
6       public static void main(String[] args)
7       {
8
9           int i=10;
            i.
11
12      }
13  }
14
```

If we wanna treat java like pure object oriented language, for primitives also we have to define classes.
That's why java has provided Wrapper classes.
Wrapper clases are availabale for every data types.



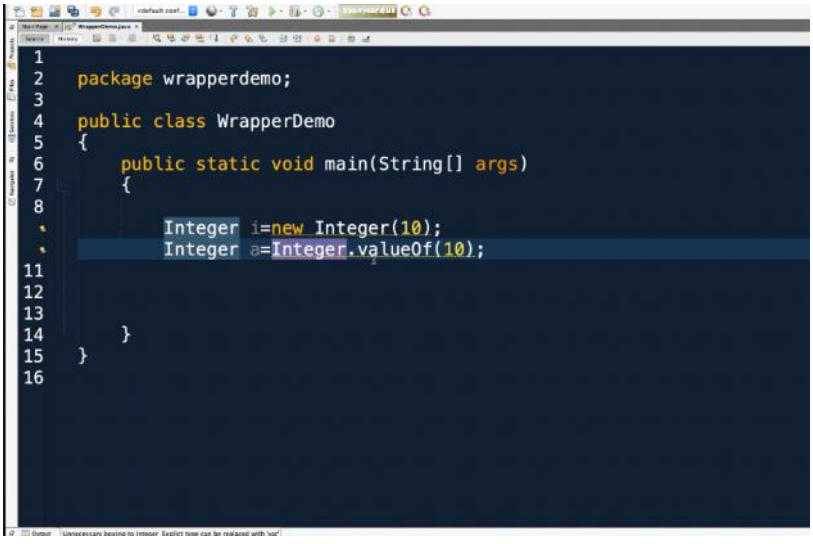
From here also we can see Object is the parent class of all other classes.

Methods of a Number class:

Method Summary

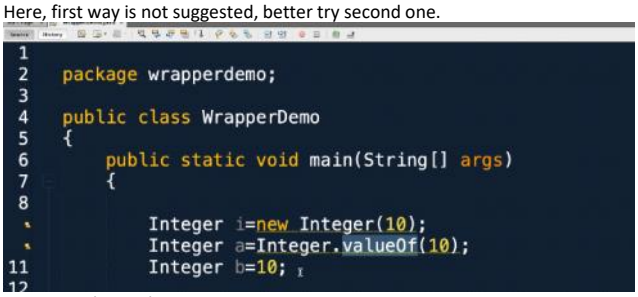
All Methods	Instance Methods	Abstract Methods	Concrete Methods

Modifier and Type	Method
byte	byteValue()
abstract double	doubleValue()
abstract float	floatValue()
abstract int	intValue()
abstract long	longValue()
short	shortValue()



```
1
2   package wrapperdemo;
3
4   public class WrapperDemo
5   {
6       public static void main(String[] args)
7       {
8
            Integer i=new Integer(10);
            Integer a=Integer.valueOf(10);
11
12
13
14      }
15  }
16
```

Here, first way is not suggested, better try second one.



```
1
2   package wrapperdemo;
3
4   public class WrapperDemo
5   {
6       public static void main(String[] args)
7       {
8
            Integer i=new Integer(10);
            Integer a=Integer.valueOf(10);
11          Integer b=10;
12
```

Can use 3rd way also.

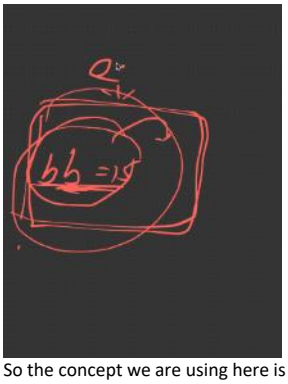
```java
package wrapperdemo;

public class WrapperDemo
{
    public static void main(String[] args)
    {

        Integer i=new Integer(10);
        Integer a=Integer.valueOf(10);
        Integer b=10;

        Byte c=15;
        Byte d=Byte.valueOf("15");
        byte bb=15;
        Byte e=Byte.valueOf(bb);

        Short f=Short.valueOf("123");

        Float g=


```

Here every variables is object

Writing direct values is also using the valueOf() method only behind the scenes.

```java
Float g=12.3f;

Float h=Float.valueOf("123.5");

Double j=Double.valueOf(123.456);
```

```java
Double j=Double.valueOf(123.456);

Character k=Character.valueOf('A');

Boolean l=Boolean.valueOf("true");
```



So the concept we are using here is boxing/wrapping where the reference is boxing around a primitive.

Now see what's happenin here

```java
Float h=Float.valueOf("123.5");
float x=h.floatValue();
float y=h;
```

Both 2 and 3 line is same because floatValue() will be called despite.
3rd line is auto-unboxing
This is unboxing, where object is converted back to primitive.

```java
int m=10;
Integer n=m;
```

Here, this is auto-boxing.

```
int m=10;
Integer n=Integer.valueOf(m);
```

Here, this is boxing.
We are putting a wrapper around m.

Integer Class

```
1
2   package wrapperdemo;
3
4   public class WrapperDemo
5   {
6       public static void main(String[] args)
7       {
8           int m1=15;
9
10          Integer m2=m1;
11
12          System.out.println(m2.equals(m1));
13
14      }
15  }
16
```

```
Compiling 1 source file to /Users/abdulbari/NetBeansPro
compile:
run:
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
1
2   package wrapperdemo;
3
4   public class WrapperDemo
5   {
6       public static void main(String[] args)
7       {
8           int m1=15;
9
10          Integer m2=m1;
11          Integer m3=15;
12
13          System.out.println(m2.equals(m3));
14
15      }
16  }
17
```

```
Compiling 1 source file to /Users/abdulbari/NetBeansP
compile:
run:
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

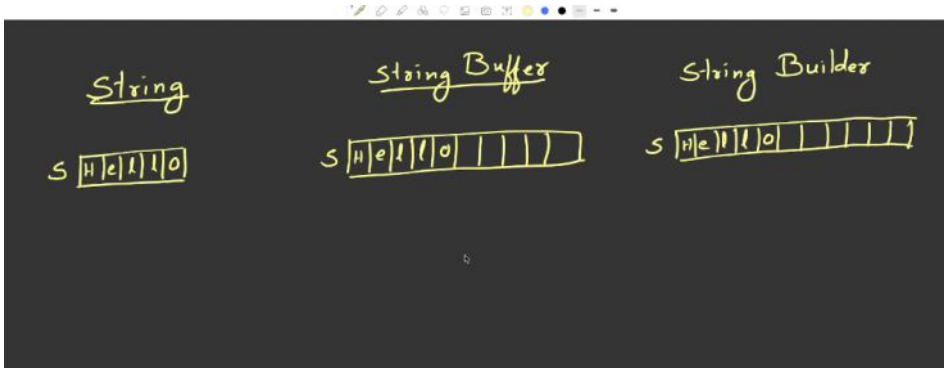
```
1
2   package wrapperdemo;
3
4   public class WrapperDemo
5   {
6       public static void main(String[] args)
7       {
8           int m1=15;
9
10          Integer m2=Integer.valueOf("123");
11          Integer m3=Integer.valueOf("1010", 2);
12
13
14          System.out.println(m3);
15
16
17      }
18  }
19
```

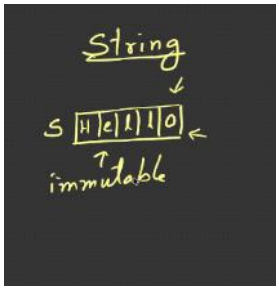
```
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/WrapperD
compile:
run:
10
BUILD SUCCESSFUL (total time: 0 seconds)
```

Because 1010 in radix 2 i.e. binary, so 10.

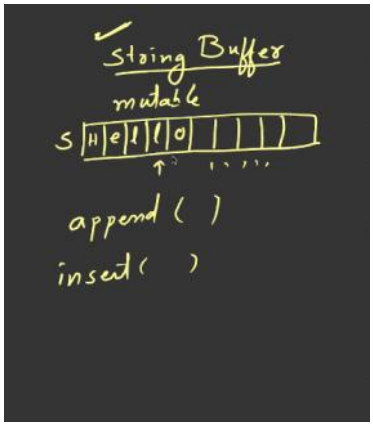
```java
package wrapperdemo;

public class WrapperDemo
{
    public static void main(String[] args)
    {
        float a=12.5f;
        Float b=-12.5f/0;


        System.out.println(b==Float.NEGATIVE_INFINITY);


    }
}
```

```
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/W
compile:
run:
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

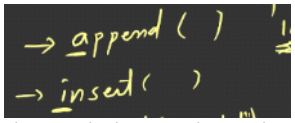




String is immutable which means if u make changes in the object, a new object will be created, changes won't be reflected on the same object.



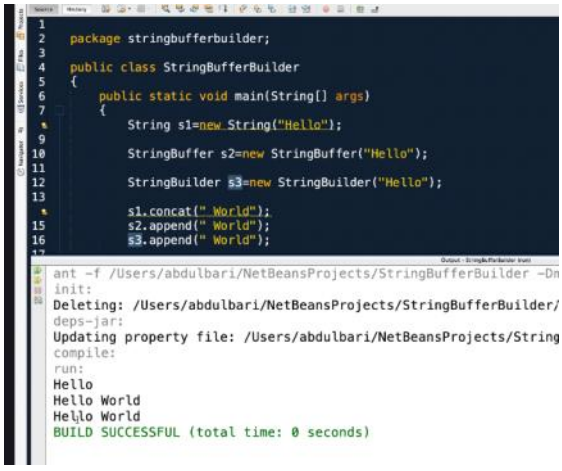
String buffer is thread safe - no more than 2 threads can accesss at the same time simultaneously.



These methods are synchronised.

Since string buffer is synchronised, its slower.

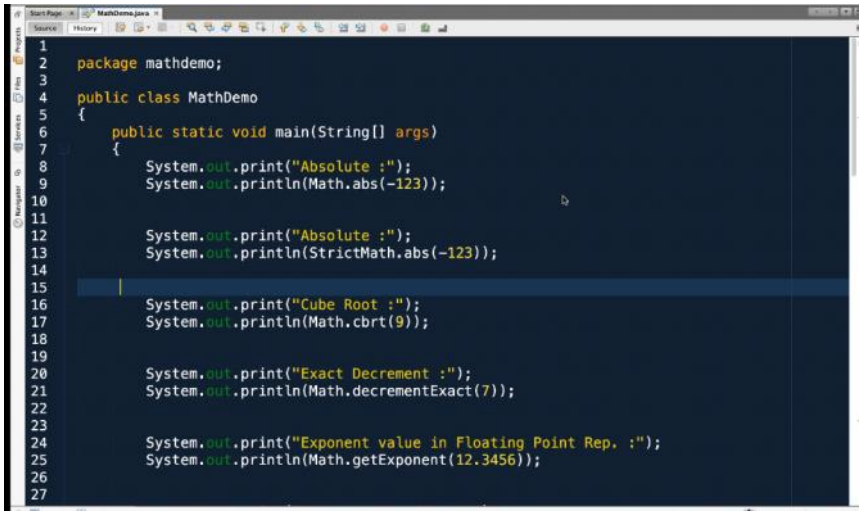
To solve it, string-builder is there.
String builder is same as string buffer, except itas not thread safe.



```
package stringbufferbuilder;

public class StringBufferBuilder
{
    public static void main(String[] args)
    {
        String s1=new String("Hello");

        StringBuffer s2=new StringBuffer("Hello");

        StringBuilder s3=new StringBuilder("Hello");

        s1.concat(" World");
        s2.append(" World");
        s3.append(" World");
```

```
ant -f /Users/abdulbari/NetBeansProjects/StringBufferBuilder -Dn
init:
Deleting: /Users/abdulbari/NetBeansProjects/StringBufferBuilder/
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/String
compile:
run:
Hello
Hello World
Hello World
BUILD SUCCESSFUL (total time: 0 seconds)
```

Math Class



```
package mathdemo;

public class MathDemo
{
    public static void main(String[] args)
    {
        System.out.print("Absolute :");
        System.out.println(Math.abs(-123));


        System.out.print("Absolute :");
        System.out.println(StrictMath.abs(-123));

        System.out.print("Cube Root :");
        System.out.println(Math.cbrt(9));


        System.out.print("Exact Decrement :");
        System.out.println(Math.decrementExact(7));


        System.out.print("Exponent value in Floating Point Rep. :");
        System.out.println(Math.getExponent(12.3456));
```

```
package mathdemo;

public class MathDemo
{
    public static void main(String[] args)
    {
        System.out.print("Absolute :");
        System.out.println(Math.abs(-123));
```

```
ant -f /Users/abdulbari/NetBeansProjects/MathDemo -Dnb.internal.action.name=ru
init:
Deleting: /Users/abdulbari/NetBeansProjects/MathDemo/build/built-jar.propertie
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/MathDemo/build/built
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/MathDemo/build/cl
compile:
run:
Absolute :123
Absolute :123
BUILD SUCCESSFUL (total time: 0 seconds)
```

Enums

```
package enumdemo;

enum Dept
{
    CS,IT,CIVIL,ECE
}

public class EnumDemo
{
    public static void main(String[] args)
    {
        Dept d=
    }
}
```

```
package enumdemo;

enum Dept
{
    CS,IT,CIVIL,EC
}

public class EnumDe
{
    public static
    {
        Dept d=Dept.
    }
}
```

enumdemo.Dept
public static final Dept CIVIL

Javadoc not found. Either Javadoc documentation for this item does not exist or you have not added specified Javadoc in the Java Platform Manager or the Library Manager.

CIVIL Dept
CS Dept
ECE Dept
IT Dept
valueOf(String name) Dept
valueOf(Class<T> enumType, String name) T
values() Dept[]
class
EnumDesc<E>

When you define an enum, it automaticallyinherits from a class called Enum.

```
package enumdemo;

enum Dept
{
    CS,IT,CIVIL,ECE
}

public class EnumDemo
{
    public static void main
    {
```

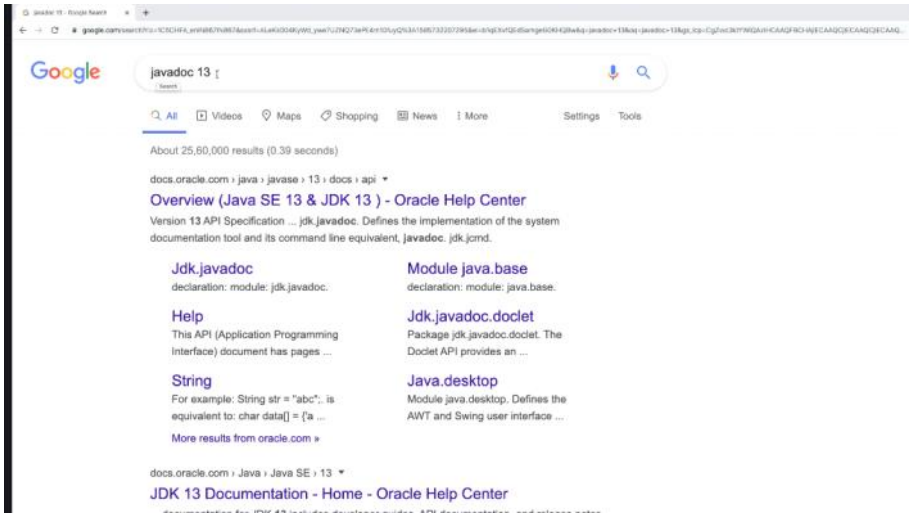
CIVIL
CS
ECE
IT
compareTo(Dept o)
describeConstable() Optional
equals(Object other)
getClass()
getDeclaringClass()
hashCode()
name()
ordinal()
toString()
notify()

```
package enumdemo;

enum Dept
{
    CS,IT,CIVIL,ECE
}

public class EnumDemo
{
    public static void main
    {
        Dept d=Dept.CS;

        System.out.println(d.);


    }
}
```

```
CIVIL
CS
ECE
IT
compareTo(Dept o)
describeConstable() Optional
equals(Object other)
getClass()
getDeclaringClass()
hashCode()
name()
ordinal()
toString()
notify()
notifyAll()
```

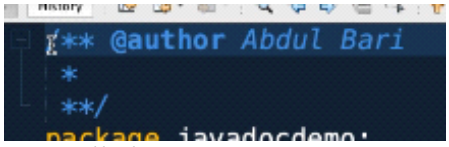
```
ant -f /Users/abdulbari/NetBeansPr
init:
Deleting: /Users/abdulbari/NetBear
deps-jar:
Updating property file: /Users/abc
Compiling 1 source file to /Users/
compile:
run:
0
BUILD SUCCESSFUL (total time: 0 se
```

Java Doc

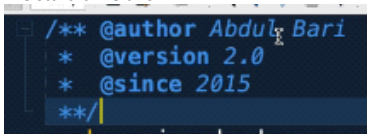


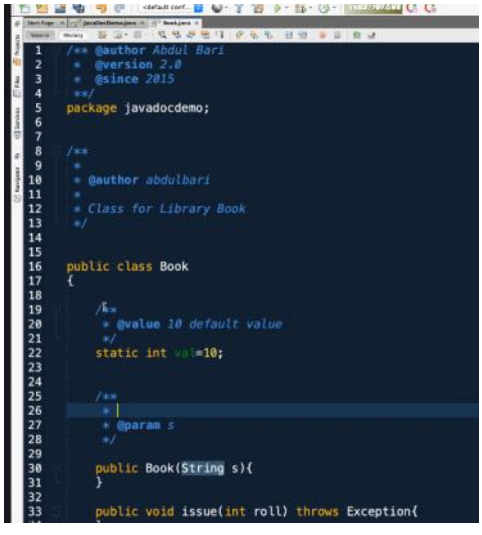
```java
/** @author Abdul Bari
 *
 **/
package javadocdemo;

public class Book
{

    static int val=10;

    public Book(String s){
    }

    public void issue(int roll) throws Exception{
    }

    public boolean available(String str){
        return true;
    }

    public String getName(int id){
        return "";
    }
}
```

Now suppose I want to document this class.
We gotta use javadoc tools.



We start it like this





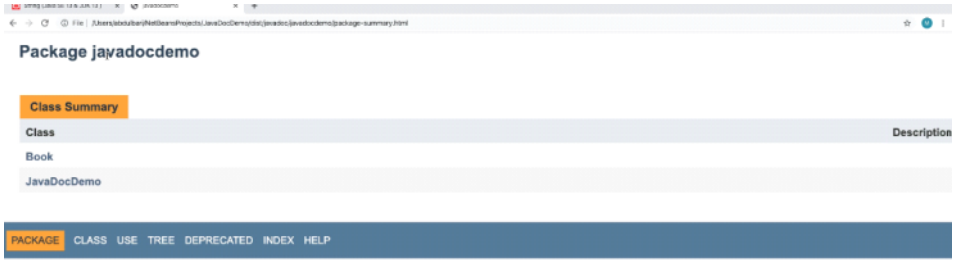
```java
    static int val=10;


    /**
     * Parametrized Constructor
     * @param s Book Name
     */

    public Book(String s){
    }

    /**
     * Issue a Book to a Student
     * @param roll roll number of a Student
     * @throws Exception if book is not available, throws Exception
     */
    public void issue(int roll) throws Exception{
    }

    public boolean available(String str){
        return true;
    }

    public String getName(int id){
        return "";
    }
}
```

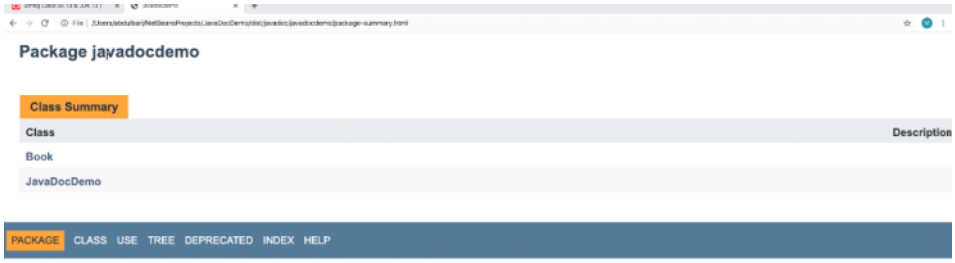
```java
    static int val=10;


    /**
     * Parametrized Constructor
     * @param s Book Name
     */

    public Book(String s){
    }

    /**
     * Issue a Book to a Student
     * @param roll roll number of a Student
     * @throws Exception if book is not available, throws Exception
     */
    public void issue(int roll) throws Exception{
    }

    /**
     * Check if book is available
     * @param str Book Name
     * @return if book is available returns true else false
     */
    public boolean available(String str){
        return true;
    }


    public String getName(int id){
        return "";
    }
}
```

Now go to - Generate Javadoc

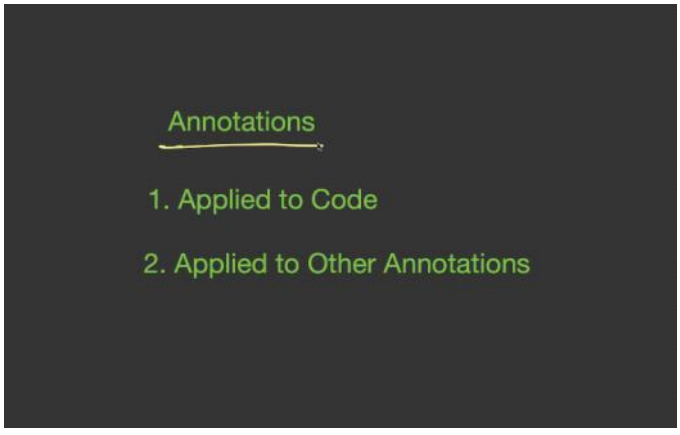

```java
/** @author Abdul Bari
 *  @version 2.0
 *  @since 2015
 **/
package javadocdemo;


/**
 *
 * @author abdulbari
 *
 * Class for Library Book
 */


public class Book
{

    /**
     * @value 10 default value
     */
    static int val=10;


    /**
     * Parametrized Constructor
     * @param s Book Name
     */

    public Book(String s){
    }
```

Now here is our doc





Annotations



Annotations are useful to provide metadata of a class.

Lets see built-in annotations

```java
package annodemo;

abstract class Parent
{
    abstract public void display();
}

class Child extends Parent
{
    public void display()
    {

    }
}

public class AnnoDemo
{
    public static void main(String[] args)
    {

    }
}
```

In this example, there is a compulsion to override display()

Now we consider an example where we won't have compulsion to override display().

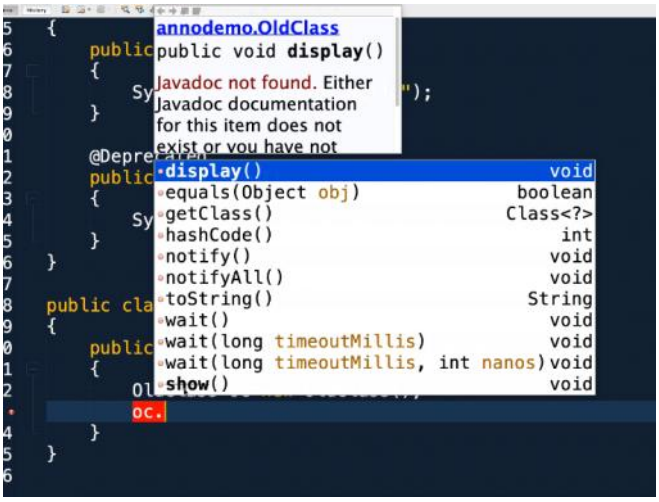
```java
package annodemo;

 class Parent
{
    public void display(){}
}

class Child extends Parent
{
    }

public class AnnoDemo
{
    public static void main(String[] args)
    {

    }
}
```

Despite there is no compulsion, suppose we want to override.
So we can use annotation here

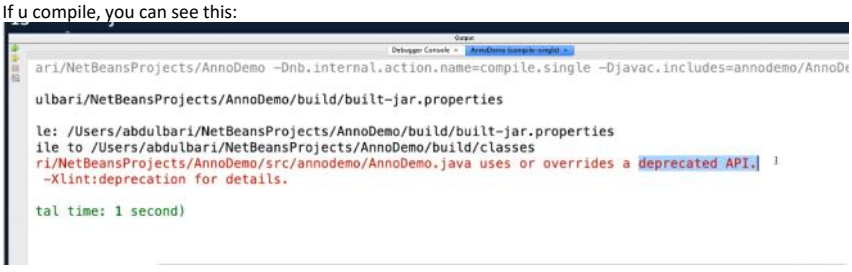
```java
package annodemo;

 class Parent
{
    public void display(){}
}

class Child extends Parent
{
    @Override
    public void display()
    {

    }

}
```

Now, suppose we wanna discard any method that won't be used in future.
We can use annotations then.

```java
{
    public void display()
    {
        System.out.println("Hello");
    }

    @Deprecated
    public void show()
    {
        System.out.println("Hi");
    }
}

public class AnnoDemo
{
    public static void main(String[] args)
    {

    }
}
```



Here we can see show() is struck off.
If u compile, you can see this:



@SuppressWarning

```java
public class AnnoDemo
{
    @SuppressWarnings("deprecation")
    public static void main(String[] args)
    {
        OldClass oc=new OldClass();
        oc.show();
    }
}
```

We saw deprecation warning before, now we wanna suppress it which has been done above.

@SafeVarargs

```java
package annodemo;

class My<T>
{
    @SafeVarargs
    private void show(T...arg)
    {
        for(T x:arg)
            System.out.println(x);
    }
```

```java
package annodemo;

class My<T>
{
    @SafeVarargs
    private void show(T...arg)
    {
        for(T x:arg)
            System.out.println(x);
    }
}
```

```
ant -f /Users/abdulbari/NetBeansProjects/AnnoDemo -Dnb.internal.action.name=compile.single -Djavac.includes=
init:
Deleting: /Users/abdulbari/NetBeansProjects/AnnoDemo/build/built-jar.properties
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/AnnoDemo/build/built-jar.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/AnnoDemo/build/classes
compile-single:
BUILD SUCCESSFUL (total time: 0 seconds)
```

If an interface having single method, it is called functional interface.

@FunctionalInterface

```java
package annodemo;

@FunctionalInterface
interface My
{
    public void show();
    public int add(int x,int y);
}

public class AnnoDemo
{
    public static void main(String[] args)
    {

    }
}
```

Its showing error since we using 2 methods - it's been taken cared off by the annotation.

User-defined annotation

Defining an annotation is similar to defining an interface.
But it should start with @.

```java
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{

}

public class AnnoDemo
{

    public static void main(String[] args)
    {
        int x;

    }
}
```

Annotation contains meta-data.
Annotations can be used at class-level, method-level.

```java
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{
}

@MyAnno
public class AnnoDemo
{
    @MyAnno
    public static void main(String[] args)
    {
        int x;

    }
}
```

Annotations can be written for parameters also.

```java
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{
}

@MyAnno
public class AnnoDemo
{
    @MyAnno
    public static void main(@MyAnno String[] args)
    {
        int x;

    }
}
```

We can annotations around local variable also.

```java
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{
}

@MyAnno
public class AnnoDemo
{
    @MyAnno
    public static void main(@MyAnno String[] args)
    {
        @MyAnno
        int x;

    }
}
```

Give it around instance variables also.

```java
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{
}

@MyAnno
public class AnnoDemo
{
    @MyAnno
    int data;
```

```java
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{
}

@MyAnno
public class AnnoDemo
{
    @MyAnno
    int data;

    @MyAnno
    public static void main(@MyAnno String[] args)
    {
        @MyAnno
        int x;

    }
}
```

Now we definin meta-data.

```java
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{
    String name();
}

@MyAnno
public class AnnoDemo
{
    @MyAnno
    int data;

    @MyAnno
    public static void main(@MyAnno String[] args)
    {
        @MyAnno
        int x;
```

Look all of em gave error because we gotta pass the metadata also.

```java
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{
    String name();
}

@MyAnno(name="Ajay")
public class AnnoDemo
{
    @MyAnno(name="Ajay")
    int data;

    @MyAnno(name="Ajay")
    public static void main(@MyAnno(name="Ajay") String[] args)
    {
        @MyAnno(name="Ajay")
        int x;

    }
```

```java
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{
    String name();
    String project();
    String date();
    String version();

}

@MyAnno(name="Ajay",project="Bank",date="1/1/2000",version="13")
public class AnnoDemo
{
    int data;

    public static void main(String[] args)
    {
        int x;
```

We can give a default value also.

```java
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{
    String name();
    String project();
    String date();
    String version() default "13";

}

@MyAnno(name="Ajay",project="Bank",date="1/1/2000")
public class AnnoDemo
{
    int data;

    public static void main(String[] args)
    {
        int x;
```

Built-in Annotations #2

These annotations are useful upon user-defined annotations.

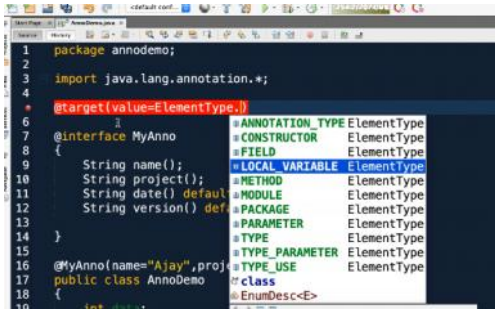
```java
package annodemo;

import java.lang.annotation.Annotation;

@interface MyAnno
{
    String name();
    String project();
    String date() default "today";
    String version() default "13";

}

@MyAnno(name="Ajay",project="Bank")
public class AnnoDemo
{
    int data;

    public static void main(String[] args)
    {
        int x;

    }
}
```

@Retention

```java
package annodemo;

import java.lang.annotation.*;


@Retention(RetentionPolicy.CLASS)

@interface MyAnno
{
    String name();
    String project();
    String date() default "today";
    String version() default "13";

}

@MyAnno(name="Ajay",project="Bank")
public class AnnoDemo
{
    int data;

    public static void main(String[] args)
    {
        int x;

    }
}
```

Here we have to mention retention policy.
If retention policy is CLASS, By using reflection, we can extract metadata

@Documented




```java
package annodemo;

import java.lang.annotation.*;

@Target(value=ElementType.LOCAL_VARIABLE)

@interface MyAnno
{
    String name();
    String project();
    String date() default "today";
    String version() default "13";

}

public class AnnoDemo
{
    int data;

    public static void main(String[] args)
    {

        int x;

    }
}
```

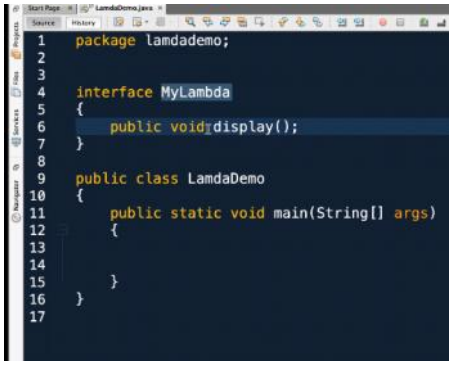
In this target annotation, we used local variable, so our annotation can only work for local variable, not

class.

Lambda Expression

Lambda expressions are used for defining anonymous methods (nameless).
It's used for interfaces

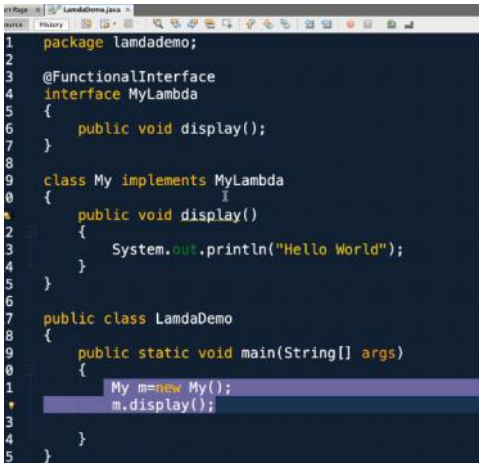
Lets define an interface now



```
package lamdademo;

interface MyLambda
{
    public void display();
}

public class LamdaDemo
{
    public static void main(String[] args)
    {


    }
}
```

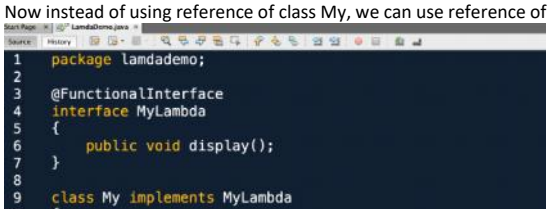
Functional interface (only one abstract method) is compulsory for lambda expressions.
Now ima define a class which implements our interface.

```
package lamdademo;

@FunctionalInterface
interface MyLambda
{
    public void display();
}

class My implements MyLambda
{
    public void display()
    {
        System.out.println("Hello World");
    }
}
```



```
package lamdademo;

@FunctionalInterface
interface MyLambda
{
    public void display();
}

class My implements MyLambda
{
    public void display()
    {
        System.out.println("Hello World");
    }
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
        My m=new My();
        m.display();

    }
}
```

Now instead of using reference of class My, we can use reference of interface MyLambda.



```
package lamdademo;

@FunctionalInterface
interface MyLambda
{
    public void display();
}

class My implements MyLambda
```

```java
package lamdademo;

@FunctionalInterface
interface MyLambda
{
    public void display();
}

class My implements MyLambda
{
    public void display()
    {
        System.out.println("Hello World");
    }
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
        MyLambda m=new My();
        m.display();

    }
}
```

```
ant -f /Users/abdulbari/NetBeansProjects/LamdaDemo -Dnb.internal.a
init:
Deleting: /Users/abdulbari/NetBeansProjects/LamdaDemo/build/built-
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/LamdaDem
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/Lamda
compile:
run:
Hello World
BUILD SUCCESSFUL (total time: 0 seconds)
```

Now instead of writing a separate class My, we can write an inner class.

Now we are using anonymous inner class. We removed class My.

```java
public class LamdaDemo
{
    public static void main(String[] args)
    {
        MyLambda m=new MyLambda(){
            public void display()
            {
                System.out.println("Hello World");
            }
        };

        m.display();

    }
}
```

```java
@FunctionalInterface
interface MyLambda
{
    public void display();
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
        MyLambda m=new MyLambda(){
            public void display()
            {
                System.out.println("Hello World");
            }
        };

        m.display();

    }
}
```

Ok, now we will be converting this into lambda expression.
Now we don't have to give the name of this method.
Also remove the interface name.

```java
@FunctionalInterface
interface MyLambda
{
    public void display();
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
        MyLambda m=
            ()->
            {
                System.out.println("Hello World");
            };

        m.display();

    }
}
```

```java
@FunctionalInterface
interface MyLambda
{
    public void display();
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
        MyLambda m=()->{System.out.println("Hello World");};

        m.display();

    }
}
```

Parameters in Lambda Expression

```java
package lamdademo;

@FunctionalInterface
interface MyLambda
{
    public void display(String str);
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
        MyLambda m=(s)->{System.out.println(s);};


        m.display("Hello World");

    }
}
```

```
ant -f /Users/abdulbari/NetBeansProjects/LamdaDemo
init:
Deleting: /Users/abdulbari/NetBeansProjects/LamdaD
deps-jar:
Updating property file: /Users/abdulbari/NetBeansP
Compiling 1 source file to /Users/abdulbari/NetBea
compile:
run:
Hello World
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
1  package lamdademo;
2
3  @FunctionalInterface
4  interface MyLambda
5  {
6      public int add(int x,int y);
7  }
8
9  public class LamdaDemo
10 {
11     public static void main(String[] args)
12     {
13         MyLambda m=(a,b)->{return a+b;};
14
15         System.out.println(m.add(20,30));
16
17     }
18 }
19
```

```
ant -f /Users/abdulbari/NetB
init:
Deleting: /Users/abdulbari/M
deps-jar:
Updating property file: /Use
Compiling 1 source file to /
compile:
run:
50
BUILD SUCCESSFUL (total time
```

```
1  package lamdademo;
2
3  @FunctionalInterface
4  interface MyLambda
5  {
6      public int add(int x,int y);
7  }
8
9  public class LamdaDemo
10 {
11     public static void main(String[] args)
12     {
13         MyLambda m=(a,b)->a+b;
14         int r=m.add(20,30);
15         System.out.println(r);
16
17     }
18 }
19
```

Capture in Lambda Expression

We wanna explore lambda with non-static methods, so took method1() below.

```
3
4  interface MyLambda
```

```java
interface MyLambda
{
    public void display();
}


class Demo
{

    public void method1()
    {


    }
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
         Demo d=new Demo();
         d.method1();
    }
}

```

Lambda expression can have local variables also.

```java
interface MyLambda
{
    public void display();
}


class Demo
{

    public void method1()
    {

        MyLambda ml=()->{
            int count=0;
            System.out.println("Hi");
            System.out.println("Bye"+count);
        };

        ml.display();
    }
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
```

Ok, now observe this code below:

```java
class Demo
{

    public void method1()
    {
        int count=0;

        MyLambda ml=()->{
            System.out.println("Hi");
            System.out.println("Bye"+count);
        };

        ml.display();

        count++;
    }
}
```

Lambda expressions can access only those variables out of its scope which are final, or effectively final.

So effectively final means, we are not doing count++ as in the above example.

```java
class Demo
{

    public void method1()
    {
        int count=0;

        MyLambda ml=()->{
            System.out.println("Hi");
            System.out.println("Bye"+count);
        };

        ml.display();

    }
}
```

Above is allowed.

I can't modify count even inside lambda expression.

```
class Demo
{

    public void method1()
    {
        int count=0;

        MyLambda ml=()->{
            System.out.println("Hi");
            System.out.println("Bye"+(count++));
        };

        ml.display();

    }
}
```

Now, lets look at instance variable.
Yes, lamda expression can access instance variables.

```
class Demo
{
    int [illegible]=10;

    public void method1()
    {
        int count=0;

        MyLambda ml=()->{
            System.out.println("Hi");
            System.out.println("Bye"+([illegible]));
        };

        ml.display();

    }
}
```

We can even modify instance variables inside lambda expression. They need not be final variables.

```
class Demo
{
    int [illegible]=10;

    public void method1()
    {
        int count=0;

        MyLambda ml=()->{
            System.out.println("Hi");
            System.out.println("Bye"+(++[illegible]));
        };

        ml.display();

    }
}
```

Can we pass lambda expressions as a parameter?

```
1   package lamdademo;
2
3
4   interface MyLambda
5   {
6       public void display();
7   }
8
9
    class UseLambda
11  {
12      public void callLambda(MyLambda ml)
13      {
14          ml.display();
15      }
16  }
17
18
19
```

```
class UseLambda
{
    public void callLambda(MyLambda ml)
    {
        ml.display();
    }
}

class Demo
{


    public void method1()
    {
        UseLambda ul=new UseLambda();
        ul.callLambda(ml);

    }
}
```

```
class UseLambda
{
    public void callLambda(MyLambda ml)
    {
        ml.display();
    }
}

class Demo
{


    public void method1()
    {
        UseLambda ul=new UseLambda();
        ul.callLambda(()->{System.out.println("Hello");});

    }
}

public class LamdaDemo
{
    public static void main(String[] args)
    {
         Demo d=new Demo();
         d.method1();
    }
```

```
Debugger Console
ant -f /Users/abdulbari/NetBeansProjects/LamdaDem
init:
Deleting: /Users/abdulbari/NetBeansProjects/Lamda
deps-jar:
Updating property file: /Users/abdulbari/NetBeans
Compiling 1 source file to /Users/abdulbari/NetBe
compile:
run:
Hello
BUILD SUCCESSFUL (total time: 0 seconds)
```

So we can pass lambda expression as parameter or object.

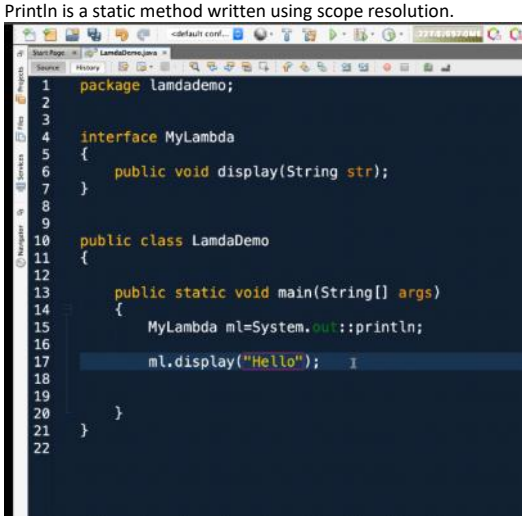
Method Reference

Method refernce is also created using functiional interface.

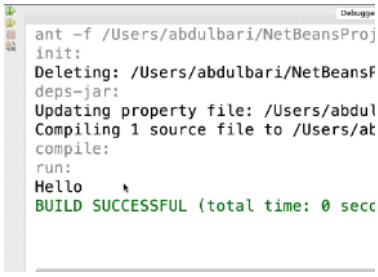
```
1  package lamdademo;
2
3
4  interface MyLambda
5  {
6      public void display(String str);
7  }
8
9
10 public class LamdaDemo
11 {
12
13     public static void main(String[] args)
14     {
15         MyLambda ml=System.out::println;
16
17
18     }
19 }
20
```

Here println will be assigned to this display() method

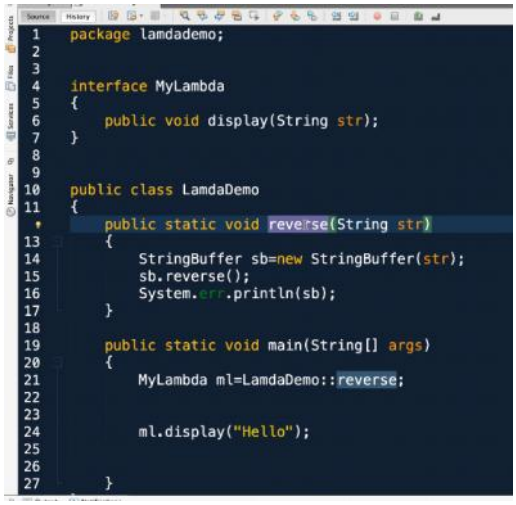
Display() will act like a println().
So the string I passed will be printed.
PrintIn is a static method written using scope resolution.



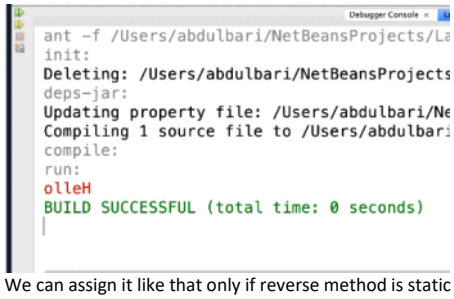
```
1  package lamdademo;
2
3
4  interface MyLambda
5  {
6      public void display(String str);
7  }
8
9
10 public class LamdaDemo
11 {
12
13     public static void main(String[] args)
14     {
15         MyLambda ml=System.out::println;
16
17         ml.display("Hello");
18
19
20     }
21 }
22
```



```
ant -f /Users/abdulbari/NetBeansProj
init:
Deleting: /Users/abdulbari/NetBeansF
deps-jar:
Updating property file: /Users/abdul
Compiling 1 source file to /Users/ab
compile:
run:
Hello
BUILD SUCCESSFUL (total time: 0 secc
```



```
1  package lamdademo;
2
3
4  interface MyLambda
5  {
6      public void display(String str);
7  }
8
9
10 public class LamdaDemo
11 {
12     public static void reverse(String str)
13     {
14         StringBuffer sb=new StringBuffer(str);
15         sb.reverse();
16         System.err.println(sb);
17     }
18
19     public static void main(String[] args)
20     {
21         MyLambda ml=LamdaDemo::reverse;
22
23
24         ml.display("Hello");
25
26
27     }
```



```
ant -f /Users/abdulbari/NetBeansProjects/La
init:
Deleting: /Users/abdulbari/NetBeansProjects
deps-jar:
Updating property file: /Users/abdulbari/Ne
Compiling 1 source file to /Users/abdulbari
compile:
run:
olleH
BUILD SUCCESSFUL (total time: 0 seconds)
```

We can assign it like that only if reverse method is static

```java
package lamdademo;


interface MyLambda
{
    public void display(String str);
}


public class LamdaDemo
{
    public  void reverse(String str)
    {
        StringBuffer sb=new StringBuffer(str);
        sb.reverse();
        System.out.println(sb);
    }

    public static void main(String[] args)
    {

        MyLambda ml=LamdaDemo::reverse;


        ml.display("Hello");

```

Here we removed static from reverse() and it's giving errors.

So now we have to create object of the LamdaDemo class.

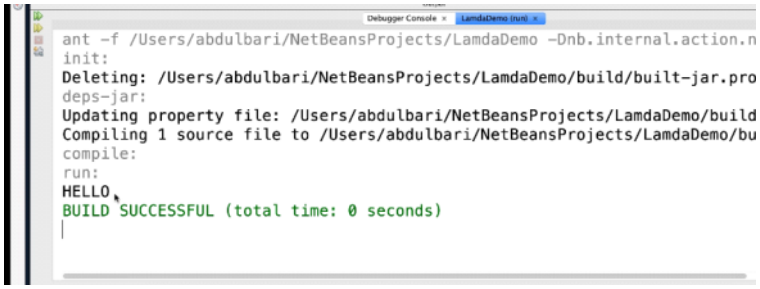
```java
package lamdademo;


interface MyLambda
{
    public void display(String str);
}


public class LamdaDemo
{
    public  void reverse(String str)
    {
        StringBuffer sb=new StringBuffer(str);
        sb.reverse();
        System.out.println(sb);
    }

    public static void main(String[] args)
    {
        LamdaDemo ld=new LamdaDemo();

        MyLambda ml=ld::reverse;


        ml.display("Hello");
```

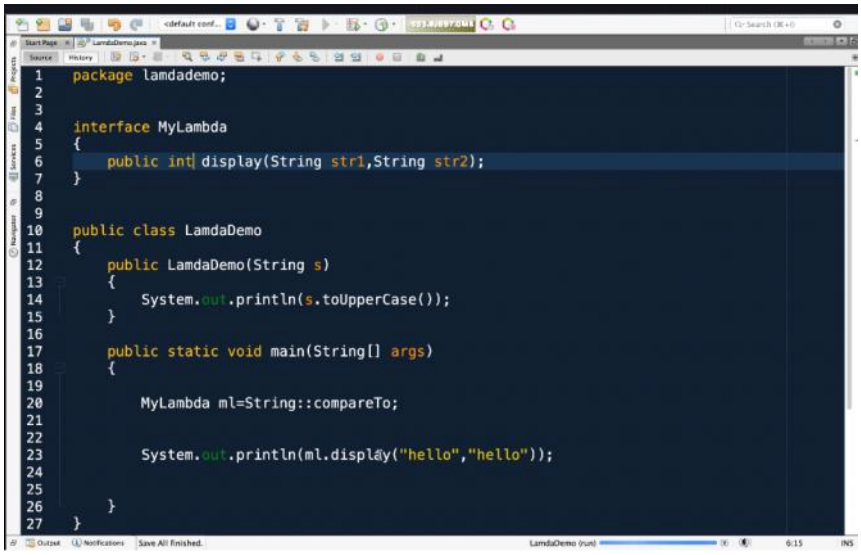
Can we assign any constructor as method reference?

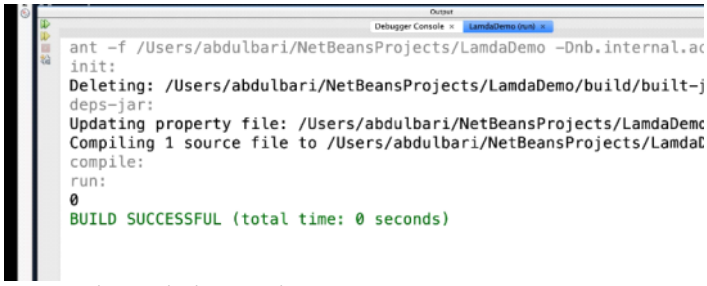
Yes -> Using new in case of constructor

```java
package lamdademo;


interface MyLambda
{
    public void display(String str);
}


public class LamdaDemo
{
    public LamdaDemo(String s)
    {
        System.out.println(s.toUpperCase());
    }

    public static void main(String[] args)
    {

        MyLambda ml=LamdaDemo::new;


        ml.display("hello");


    }
}
```

```
ant -f /Users/abdulbari/NetBeansProjects/LamdaDemo -Dnb.internal.action.n
init:
Deleting: /Users/abdulbari/NetBeansProjects/LamdaDemo/build/built-jar.pro
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/LamdaDemo/build
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/LamdaDemo/bu
compile:
run:
HELLO
```

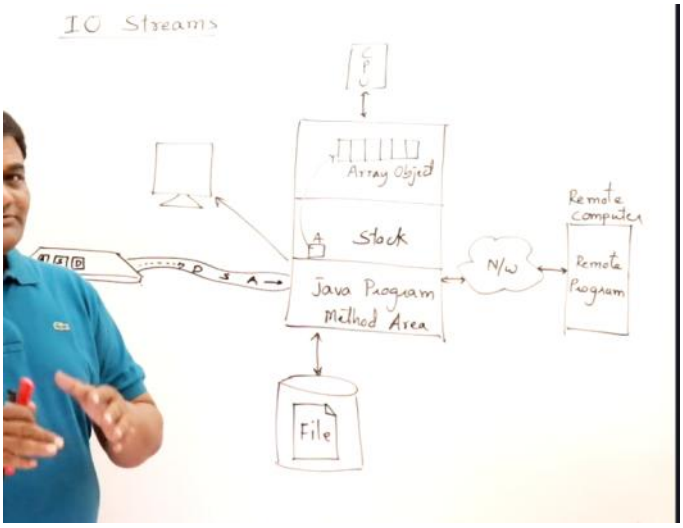


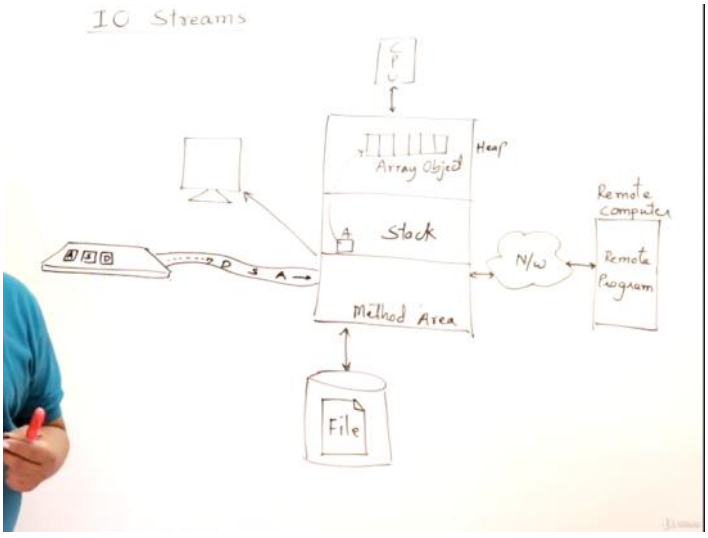




Output is 0 because both are equal.

Java IO Streams





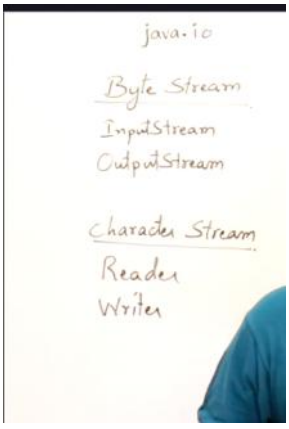
Stack+method area are only part of the program. Rest all are resources.

Program talks with all those resources by transferring the data.

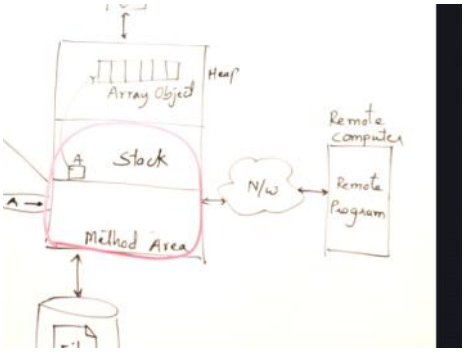
Input output can be seen here all over from keyboard, monitor etc.

Stream is a flow of data.

Now, data may be flowing from resource to a program, or from program to resource.



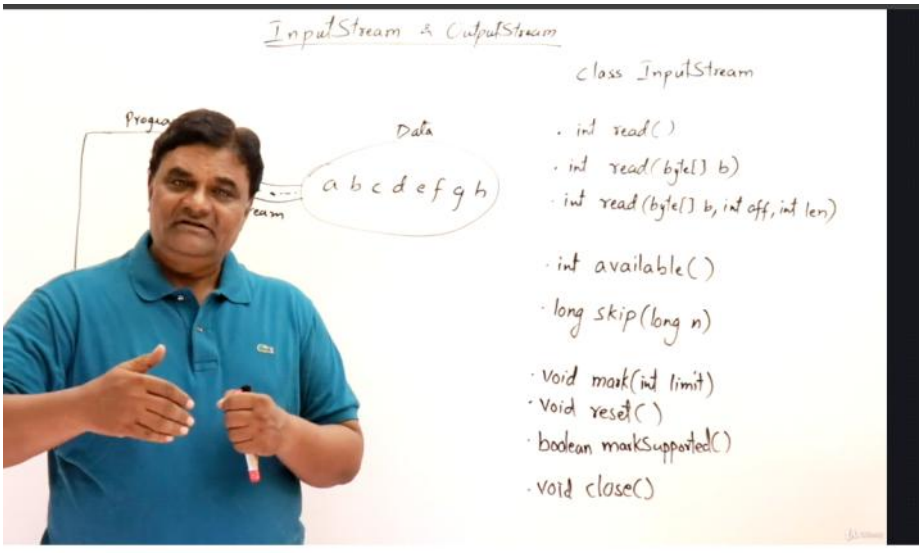
Ok now what is buffer.



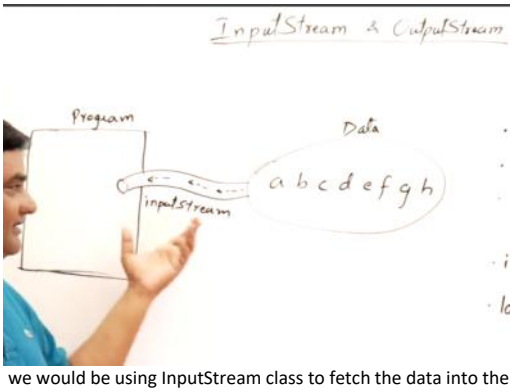
Suppose computer is sending the data ,and remote computer is receiving the data, But the speed of sending and receiving of the 2 computers might not be the same. Since there is no match in speed, we need some buffer.

Now, if we wanna send a file, it will be sent byte by byte.
Suppose we wanna send "John", then it would go like - 'J', 'O', 'H', 'N'.

InputStream and OutputStream



InputStream and OutputStream -> these are the base classes



we would be using InputStream class to fetch the data into the program.

Read() will read 1 byte from the resource i.e. only 'a' will come into the program. Now when you read the data, Its no longer available in the stream, its in your program now.

Since we have 8 alphabets here, we have to call read() for 8 times

If it has gone to the end of the stream i.e. got nothing to read, it will return -1.

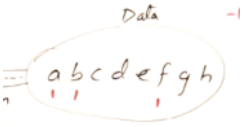
Now in read(byte[] b) -> it will read some characters from stream based upon array size.
It will read collection of bytes.

Now suppose we don't wanna start filling the byte array from beginning, we can usw this:

- int read(byte[] b, int off, int len)

Int available() will return no. of bytes available in the stream.

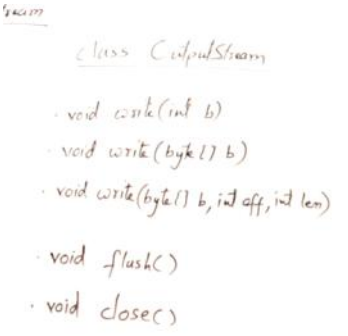
Skip()



Suppose we wanna skip one character from the stream then use it

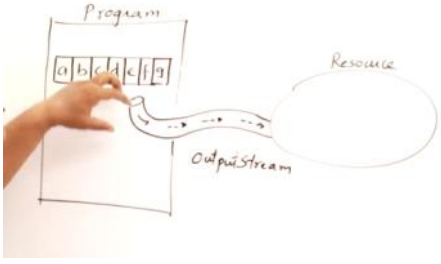
Void mark(limit)
So if we don't want our characters to be removed from the resource when reading it into our program, we use mark. If we wanna come back to the same mark, we can use reset().
After we cross the limit, the mark will be void and removed.

Mark() is only possible if it's a bufferedstream



Now lets get ahead with OutputStream

For sending a character from program to resource, we need output stream.

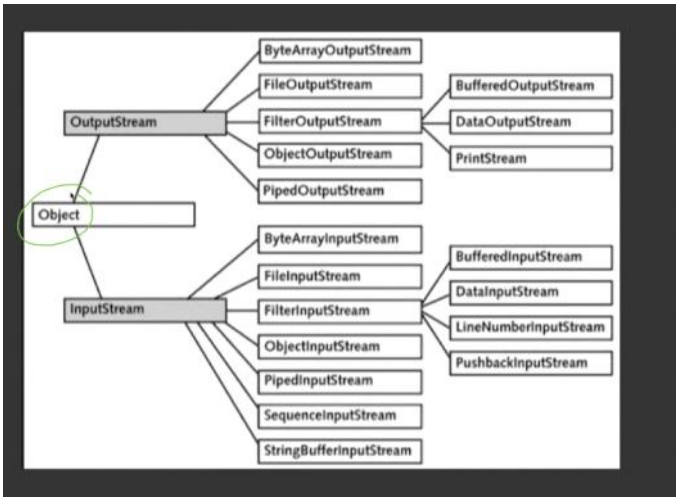


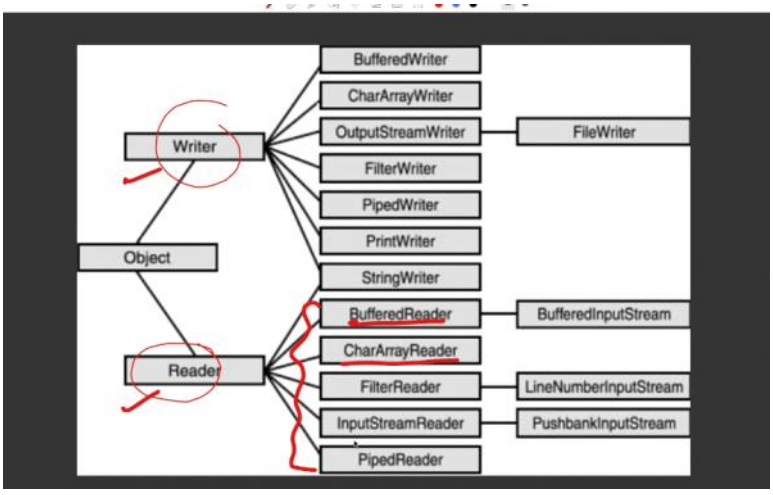
The output stream methods similar to inputstreams - just write() instead of read().

- void flush()

This flush() will work for buffered output streams.
Flush will push the data from buffer to resource

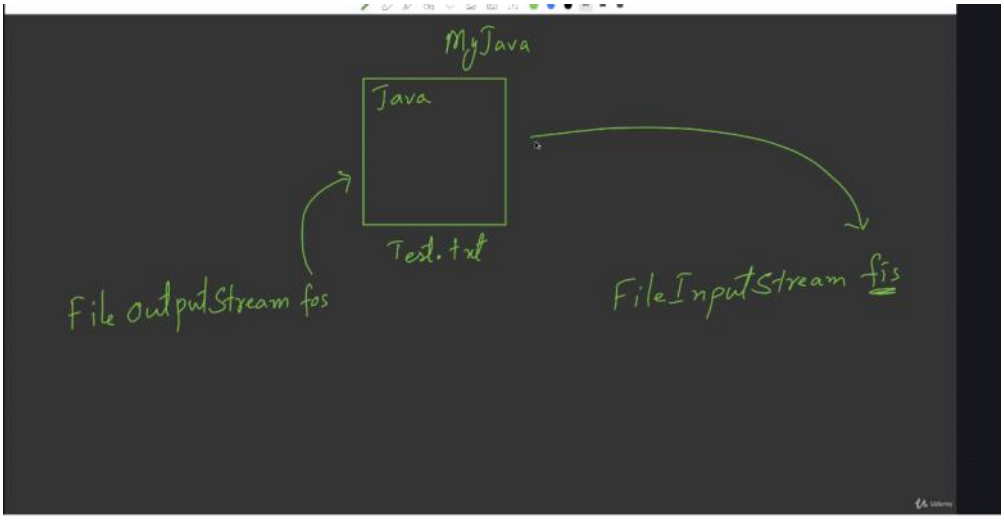
Java.io classes





File OutputStream

Now, we will create Test.txt using FileOutputStream and write some text in that, and then use FileInputStream to Read the contents of that file.



```java
package fileexample;

public class FileExample
{
    public static void main(String[] args)
    {
        FileOutputStream fos=new FileOutputStream("C:/MyJava/Test.txt")
    }
}
```

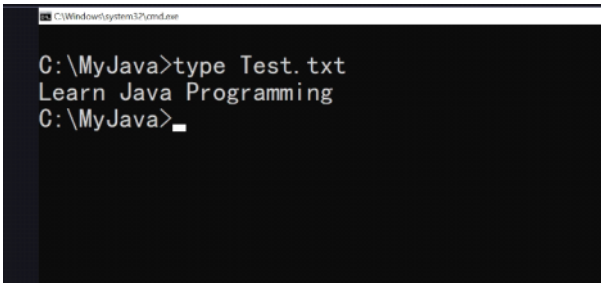
This file will be created
But for using this claSS, gotta import java.io
We also have to handle an exception.

```java
public class FileExample
{
    public static void main(String[] args)
    {
        try
        {
        FileOutputStream fos=new FileOutputStream("C:/MyJava/Test.txt");

        }
        catch(FileNotFoundException e)
        {
            System.out.println(e);
        }
    }
}
```

```java
public class FileExample
{
    public static void main(String[] args)
    {
        try
        {
        FileOutputStream fos=new FileOutputStream("C:/MyJava/Test.txt");
        String str="Lea
        fos.write(str.getBytes());
```

```java
fos.write(str.getBytes());


}
catch(FileNotFoundException e)
{
    System.out.println(e);
}
catch(IOException e)
{
    System.out.println(e);
}
```

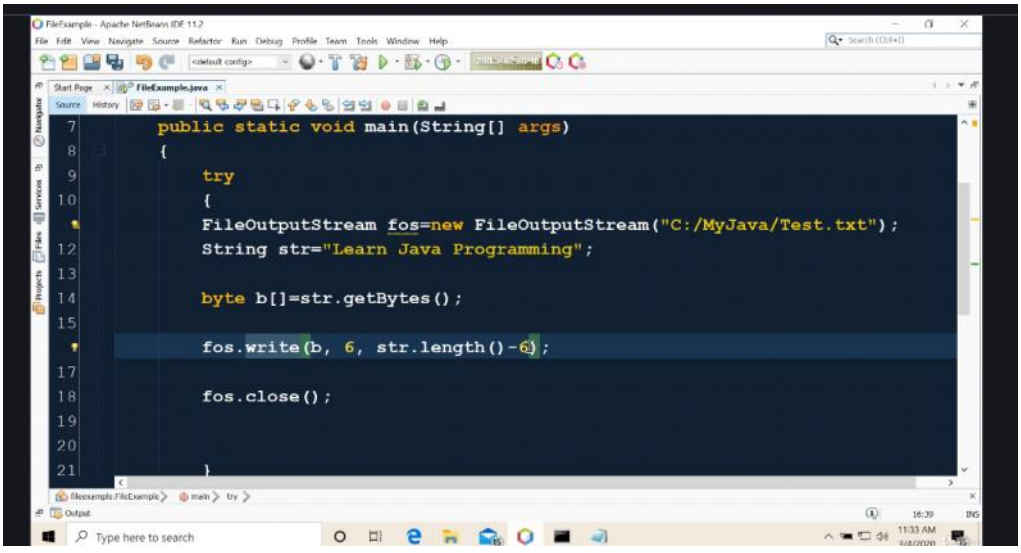
```java
    public static void main(String[] args)
    {
        try
        {
        FileOutputStream fos=new FileOutputStream("C:/MyJava/Test.txt");
        String str="Learn Java Programming";

        fos.write(str.getBytes());

        fos.close();


        }
        catch(FileNotFoundException e)
        {
```

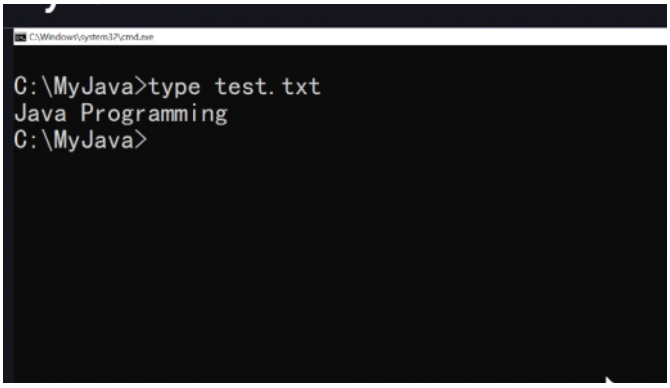


Suppose we want to write the characters byte by byte instead of in once.

```
public static void main(String[] args)
{
    try
    {
    FileOutputStream fos=new FileOutputStream("C:/MyJava/Test.txt");
    String str="Learn Java Programming";

    byte b[]=str.getBytes();

    for(byte x:b)
        fos.write(x);

    fos.close();
```

Same will be printed.





Try with resources

```
public static void main(String[] args) throws Exception
{

 try (FileOutputStream fos = new FileOutputStream("C:/MyJava/Test.txt"))
  String str="Learn Java Programming";

  byte b[]=str.getBytes();

 fos.write(b);
       }


}
```

```
C:\MyJava>type test.txt
Learn Java Programming
C:\MyJava>
```

File Input Stream and File reader

```
public static void main(String[] args) throws Exception
{

 try (FileInputStream fis = new FileInputStream("C:/MyJava/Test.txt"))
 {
     byte b[]=new byte[fis.available()];

     fis.read(b);


 }
}
}
```

```
public static void main(String[] args) throws Exception
{

 try (FileInputStream fis = new FileInputStream("C:/MyJava/Test.txt"))
 {
     byte b[]=new byte[fis.available()];

     fis.read(b);

     String str=new String(b);

     System.out.println(str);

 }
```

```
Updating property file: \\Mac\Home\Documents\NetBeansP
Compiling 1 source file to \\Mac\Home\Documents\NetBea
compile:
run:
Learn Java Programming
BUILD SUCCESSFUL (total time: 1 second)
```

```
public static void main(String[] args) throws Exception
{

 try (FileInputStream fis = new FileInputStream("C:/MyJava/Test.txt"))
 {
     int x;

     do
     {
         x=fis.read();
         System.out.print((char)x);
     }while(x!=-1);

 }
```

```
deps-jar:
Updating property file: \\Mac\Home\Documents\NetBeansProjects\
Compiling 1 source file to \\Mac\Home\Documents\NetBeansProject
compile:
run:
Learn Java ProgrammingBUILD SUCCESSFUL (total time: 0 seconds)
```

```
public static void main(String[] args) throws Exception
{

 try (FileInputStream fis = new FileInputStream("C:/MyJava/Test.txt"))
 {
     int x;

     do
     {
         x=fis.read();
         if(x!=-1)
            System.out.print((char)x);
     }while(x!=-1);
```

```
{

 try (FileInputStream fis = new FileInputStream("C:/MyJava/Test.txt"))
 {
     int x;

     while((x=fis.read())!=-1)
     {
         System.out.print((char)x);
     }

 }
}
}
```

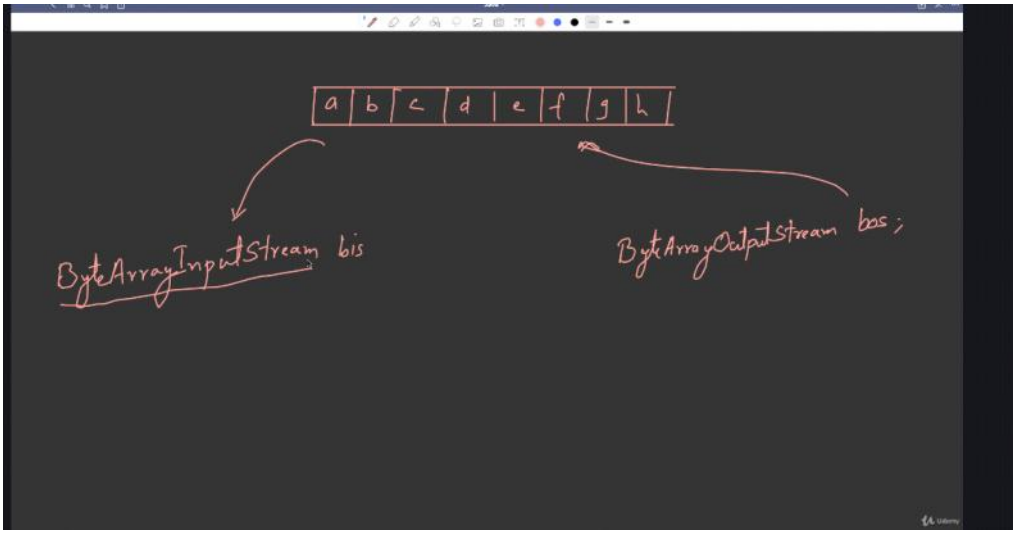
Now instead lets use File Reader

```
{

 try (FileReader fr = new FileReader("C:/MyJava/Test.txt"))
 {
     int x;


     while((x=fr.read())!=-1)
     {
         System.out.print((char)x);
     }

 }
}
```

Yes same output

```
deps-jar:
Updating property file: \\Mac\Home\Documents\NetBeansProjects\Fil
Compiling 1 source file to \\Mac\Home\Documents\NetBeansProjects\
compile:
run:
Learn Java ProgrammingBUILD SUCCESSFUL (total time: 1 second)
```

Byte Streams and Char Array Readers



Previosuly we saw File Input Stream and File Output Stream. In that, source of data was file.
But here source of data is array.
The array would be treated as streams.

Code for reading byte by byte:

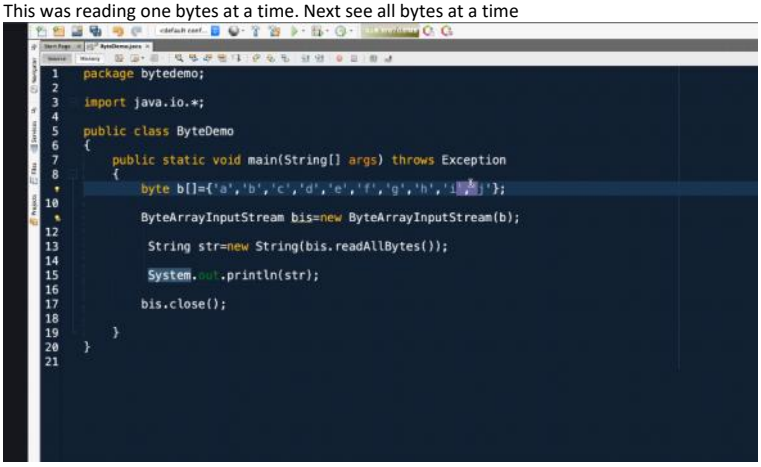
```
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {
        byte b[]={'a','b','c','d','e','f','g','h','i','j'};

        ByteArrayInputStream bis=new ByteArrayInputStream(b);

        int x;

        while((x=bis.read())!=-1)
        {
            System.out.println((char)x);
        }
        bis.close();

    }
}
```

```
a
b
c
d
e
f
g
h
i
j
BUILD SUCCESSFUL (total time: 1 second)
```

```
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {
        byte b[]={'a','b','c','d','e','f','g','h','i','j'};

        ByteArrayInputStream bis=new ByteArrayInputStream(b);

        int x;

        while((x=bis.read())!=-1)
        {
            System.out.print((char)x);
        }
```

```
ant -f /Users/abdulbari/NetBeansProjects/ByteDemo -Dnb.internal.action.name=run
init:
Deleting: /Users/abdulbari/NetBeansProjects/ByteDemo/build/built-jar.properties
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/ByteDemo/build/built-j
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/ByteDemo/build/clas
compile:
run:
abcdefghijBUILD SUCCESSFUL (total time: 0 seconds)
```

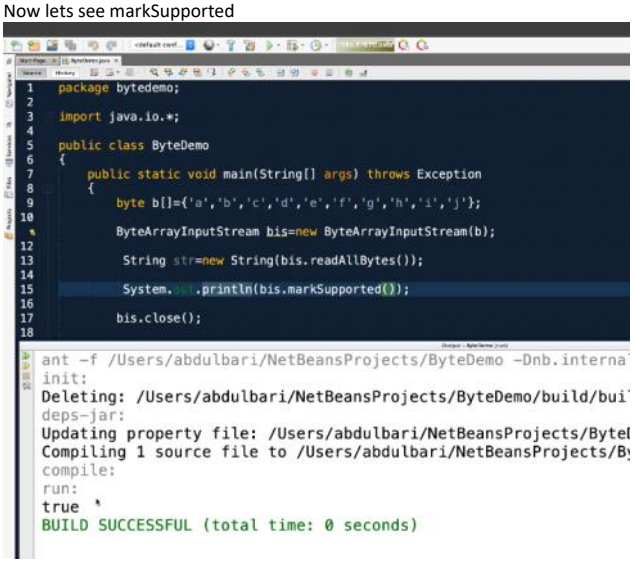
This was reading one bytes at a time. Next see all bytes at a time



```
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {
        byte b[]={'a','b','c','d','e','f','g','h','i','j'};

        ByteArrayInputStream bis=new ByteArrayInputStream(b);

         String str=new String(bis.readAllBytes());

         System.out.println(str);

        bis.close();

    }
}
```

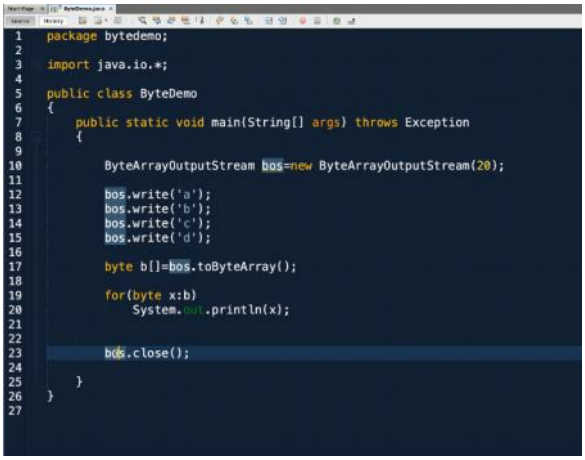
```
ant -f /Users/abdulbari/NetBeansProjects/ByteDemo -Dnb.internal.action.name=run run
init:
Deleting: /Users/abdulbari/NetBeansProjects/ByteDemo/build/built-jar.properties
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/ByteDemo/build/built-jar.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/ByteDemo/build/classes
compile:
run:
abcdefghij
```

Now lets see markSupported



```
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {
        byte b[]={'a','b','c','d','e','f','g','h','i','j'};

        ByteArrayInputStream bis=new ByteArrayInputStream(b);

         String str=new String(bis.readAllBytes());

         System.out.println(bis.markSupported());

        bis.close();

ant -f /Users/abdulbari/NetBeansProjects/ByteDemo -Dnb.interna
init:
Deleting: /Users/abdulbari/NetBeansProjects/ByteDemo/build/bui
deps-jar:
Updating property file: /Users/abdulbari/NetBeansProjects/Byte
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/By
compile:
run:
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

ByteArrayOutputStream



```
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {

        ByteArrayOutputStream bos=new ByteArrayOutputStream(20);

        bos.write('a');
        bos.write('b');
        bos.write('c');
        bos.write('d');

        byte b[]=bos.toByteArray();

        for(byte x:b)
            System.out.println(x);


        bos.close();

    }
}
```

```
        byte b[]=bos.toByteArray();
deps-jar:
Updating property file: /Users/abdulbari/Net
Compiling 1 source file to /Users/abdulbari/
compile:
run:
97
98
99
100
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
1  package bytedemo;
2
3  import java.io.*;
4
5  public class ByteDemo
6  {
7      public static void main(String[] args) throws Exception
8      {
9
10         ByteArrayOutputStream bos=new ByteArrayOutputStream(20);
11
12         bos.write('a');
13         bos.write('b');
14         bos.write('c');
15         bos.write('d');
16
17         byte b[]=bos.toByteArray();
18
19         for(byte x:b)
20             System.out.println((char)x);
21
22
23         bos.close();
24
25     }
26 }
27
```

```
deps-jar:
Updating property file: /U
Compiling 1 source file to
compile:
run:
a
b
c
d
BUILD SUCCESSFUL (total ti
```

Now suppose we want to write these into a file.

```
1  package bytedemo;
2
3  import java.io.*;
4
5  public class ByteDemo
6  {
7      public static void main(String[] args) throws Exception
8      {
9
10         ByteArrayOutputStream bos=new ByteArrayOutputStream(20);
11
12         bos.write('a');
13         bos.write('b');
14         bos.write('c');
15         bos.write('d');
16
17
18         bos.writeTo(new FileOutputStream("/Users/abdulbari/Desktop/Test.txt"));
19
20         bos.close();
21
22     }
23 }
24
```

```
abcd
```

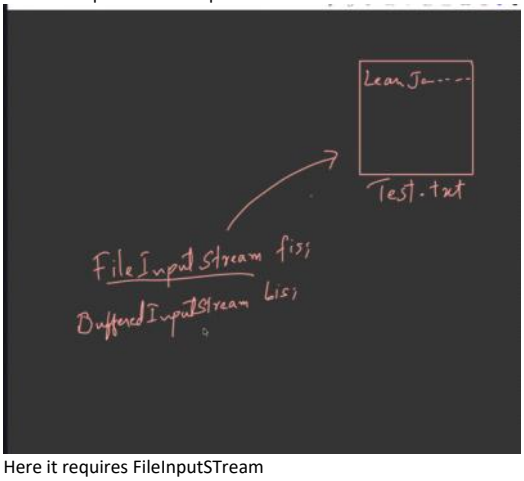
Now lets see CharArrayReader

```java
package bytedemo;

import java.io.*;

public class ByteDemo
{
    public static void main(String[] args) throws Exception
    {

        char c[]={'a','b','c','d','e','f','g','h','i','j'};

        CharArrayReader cr=new CharArrayReader(c);

        int x;

        while((x=cr.read())!=-1)
        {
            System.out.println((char)x);

        }

        cr.close();

    }
}
```

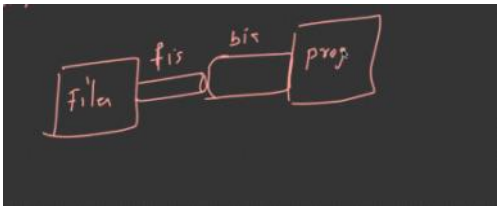
```
a
b
c
d
e
f
g
h
i
j
BUILD SUCCESSFUL
```

2x 12:05 / 1

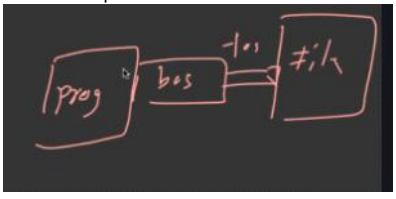
Buffered Input Stream
Buffered Input Stream requires some stream to function.



Here it requires FileInputSTream



BufferedOutputStream



```java
public class BufferedDemo
{
    public static void main(String[] args)throws Exception
    {
        FileInputStream fis=new FileInputStream("C:/MyJava/Test.txt");
        BufferedInputStream bis=new BufferedInputStream(fis);

        int x;
        while((x=bis.read())!=-1)
        {
            System.out.print((char)x);
        }
    }
}
```

```
deps-jar:
Updating property file: \\Mac\Home\Documents\NetBeansProjects\Bu
Compiling 1 source file to \\Mac\Home\Documents\NetBeansProjects
compile:
run:
Learn Java ProgrammingBUILD SUCCESSFUL (total time: 1 second)
```

But then, what is the benefit of using Buffered Input Stream?
Is markSupported() work on BufferedInputSTream?
Lets see

```java
public class BufferedDemo
{
    public static void main(String[] args)throws Exception
    {
        FileInputStream fis=new FileInputStream("C:/MyJava/Test.txt");
        BufferedInputStream bis=new BufferedInputStream(fis);

        System.out.println("File "+fis.markSupported());
        System.out.println("Buufer "+bis.markSupported());
    }
}
```

```
Compiling 1 source file to \\Mac\Home\Documents\
compile:
run:
File false
Buufer true
BUILD SUCCESSFUL (total time: 1 second)
```

We can see file is false, but buffer is true.
Because buffer stores data, while file doesn't

```java
public class BufferedDemo
{
    public static void main(String[] args)throws Exception
    {
        FileInputStream fis=new FileInputStream("C:/MyJava/Test.txt");
        BufferedInputStream bis=new BufferedInputStream(fis);

        System.out.println(bis.read());
        System.out.println(bis.read());
        System.out.println(bis.read());
        bis.mark(10);
        System.out.println(bis.read());
        System.out.println(bis.read());

    }
```

```java
    public static void main(String[] args)throws Exception
    {
        FileInputStream fis=new FileInputStream("C:/MyJava/Test.txt");
        BufferedInputStream bis=new BufferedInputStream(fis);

        System.out.println(bis.read());
        System.out.println(bis.read());
        System.out.println(bis.read());
        bis.mark(10);
        System.out.println(bis.read());
        System.out.println(bis.read());
        bis.reset();

        System.out.println(bis.read());
        System.out.println(bis.read());
```

Gotta typecast and removing ln :-

```
deps-jar:
Updating property file: \
Compiling 1 source file t
compile:
run:
LearnrnBUILD SUCCESSFUL (
```

Now lets see buffered Reader

```java
    {
        public static void main(String[] args)throws Exception
        {
            FileReader fis=new FileReader("C:/MyJava/Test.txt");
            BufferedReader bis=new BufferedReader(fis);

            System.out.print((char)bis.read());
            System.out.print((char)bis.read());
            System.out.print((char)bis.read());
            bis.mark(10);
            System.out.print((char)bis.read());
            System.out.print((char)bis.read());
            bis.reset();

            System.out.print((char)bis.read());
            System.out.print((char)bis.read());
```

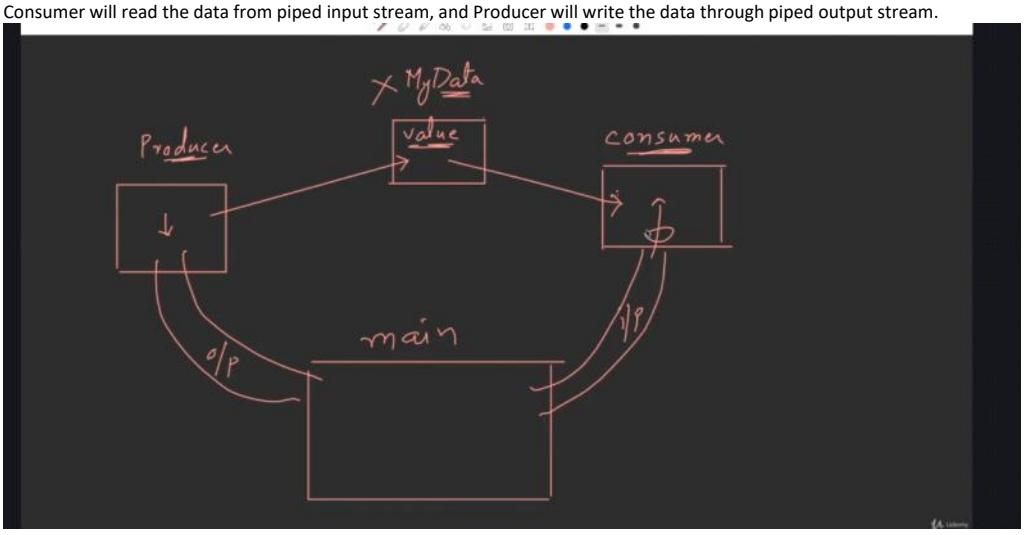
```
deps-jar:
Updating property file: \\Mac\Home\Documents\NetB
Compiling 1 source file to \\Mac\Home\Documents\N
compile:
run:
LearnrnBUILD SUCCESSFUL (total time: 1 second)
```

For reading the remaining string:

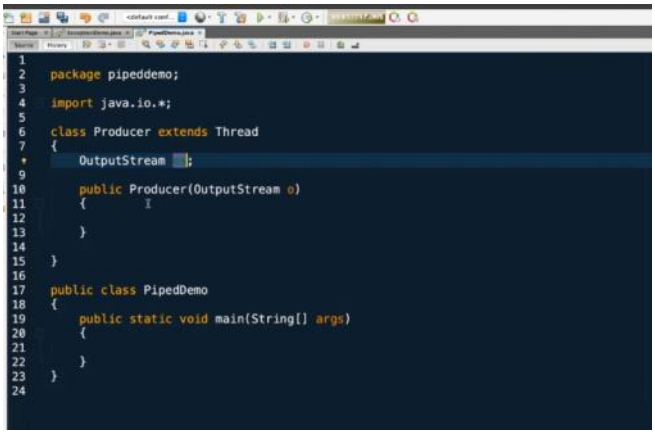
```
21
22          System.out.println("String "+bis.readLine());
Output - BufferedDemo (run)
Updating property file: \\Mac\Home\Documents\NetBeansProjects\Buff
Compiling 1 source file to \\Mac\Home\Documents\NetBeansProjects\B
compile:
run:
LearnrnString  Java Programming
BUILD SUCCESSFUL (total time: 1 second)
```

Piped Streams

Consumer will read the data from piped input stream, and Producer will write the data through piped output stream.



We will connect both pipes in the main()
We will write one thread class for Producer and another thread class for Consumer



```
1
2   package pipeddemo;
3
4   import java.io.*;
5
6   class Producer extends Thread
7   {
*       OutputStream ;
9
10      public Producer(OutputStream o)
11      {
12
13      }
14
15  }
16
17  public class PipedDemo
18  {
19      public static void main(String[] args)
20      {
21
22      }
23  }
24
```

```java
package pipeddemo;

import java.io.*;

class Producer extends Thread
{
    OutputStream os;

    public Producer(OutputStream o)
    {
        os=o;
    }

    public void run()
    {
        int count=1;

        while(true)
        {
            try{
            os.write(count);
            os.flush();

            System.out.println("Producer "+count);
            count++;
            }catch(Exception e){}
        }
    }

}

public class PipedDemo
```

```java
class Consumer extends Thread
{
    InputStream is;

    public Consumer(InputStream s)
    {
        is=s;
    }

    public void run()
    {
        int x;

        while(true)
        {
            try{

                x=is.read();

            System.out.println("Consumer "+x);
            System.out.flush();

            }catch(Exception e){}
        }
    }
```

```java
public class PipedDemo
{
    public static void main(String[] args) throws Exception
    {
        PipedInputStream pis=new PipedInputStream();
        PipedOutputStream pos=new PipedOutputStream();

        pos.connect(pis);

        Producer p=new Producer(pos);
        Consumer c=new Consumer(pis);

        p.start();

        c.start();

    }
}
```

```
Consumer 82
Consumer 83
Consumer 84
Consumer 85
Consumer 86
Consumer 87
Producer 8964
Consumer 88
Consumer 89
Consumer 90
Consumer 91
```

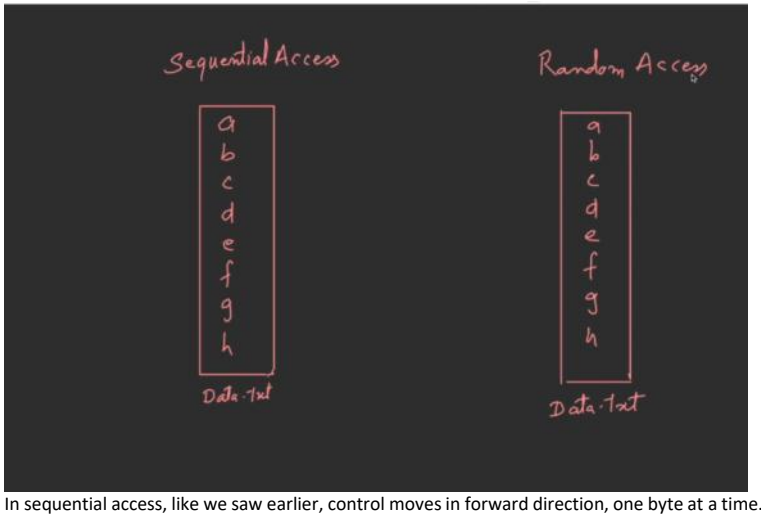
So we can see Producer is very fast while consumer is very slow. Its not synchronised.
So lets try making producer sleep.

```
class Producer extends Thread
{
    OutputStream os;

    public Producer(OutputStream o)
    {
        os=o;
    }

    public void run()
    {
        int count=1;

        while(true)
        {
            try{
              .write(count);
              .flush();

            System.out.println("Producer "+count);
            System.out.flush();

            Thread.sleep(10);
            count++;
            }catch(Exception e){}
        }
    }

}

class Consumer extends Thread
{
    InputStream is;
```

```
    public Consumer(InputStream s)
    {
        is=s;
    }

    public void run()
    {
        int x;

        while(true)
        {
            try{

                x=is.read();

            System.out.println("Consumer "+x);
            System.out.flush();
            Thread.sleep(10);

            }catch(Exception e){}
        }
    }

}
```

```
Consumer 245
Producer 246
Consumer 246
Producer 247
Consumer 247
Producer 248
Consumer 248
Producer 249
Consumer 249
BUILD STOPPED (total time: 3 seconds)
```

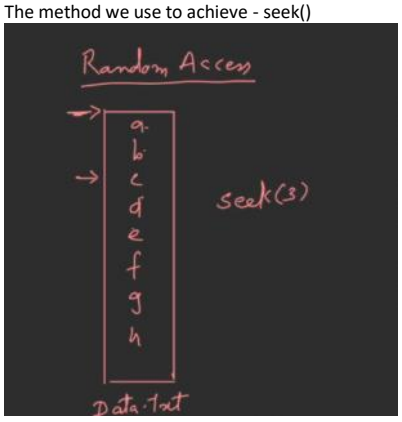
Random Access File



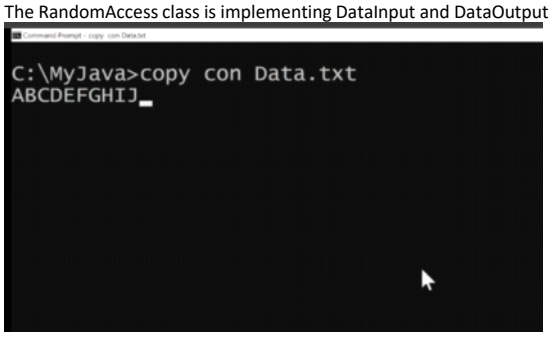
In sequential access, like we saw earlier, control moves in forward direction, one byte at a time.

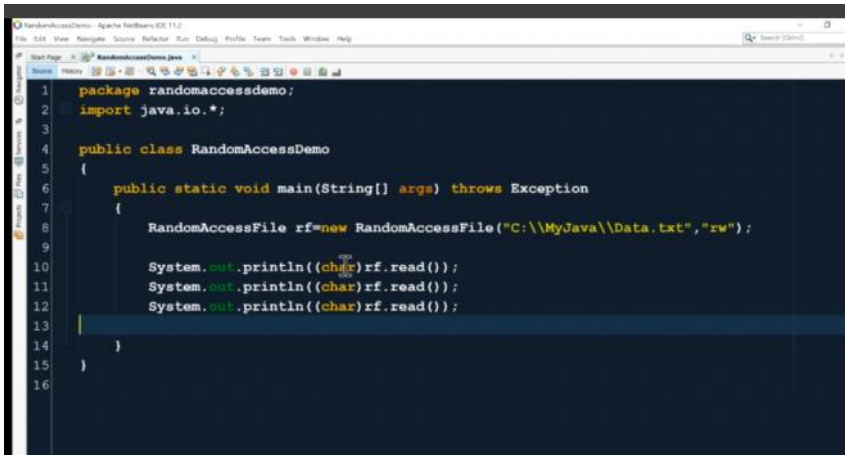
In random access, same pointer will be used both for reading and writing, unlike sequential.

In random access, we can move to any position with based on the no. of bytes from first position
The method we use to achieve - seek()

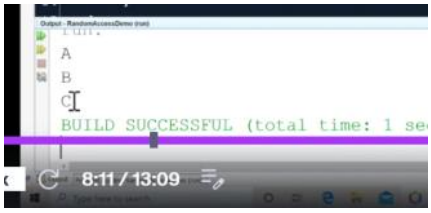


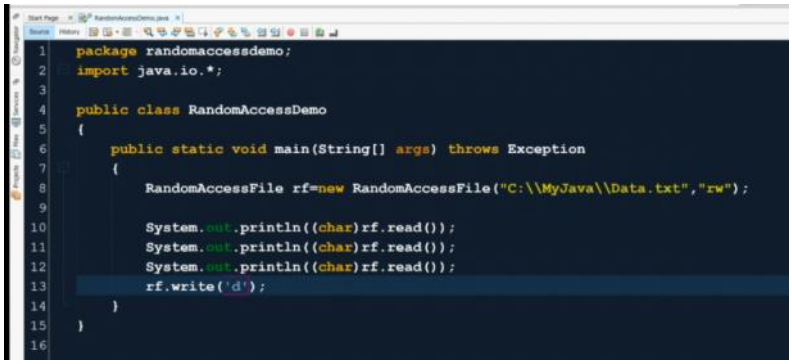
The RandomAccess class is implementing DataInput and DataOutput interfaces.



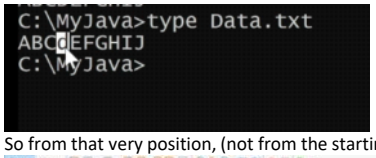


```
1  package randomaccessdemo;
2  import java.io.*;
3
4  public class RandomAccessDemo
5  {
6      public static void main(String[] args) throws Exception
7      {
8          RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","rw");
9
10         System.out.println((char)rf.read());
11         System.out.println((char)rf.read());
12         System.out.println((char)rf.read());
13
14     }
15 }
16
```



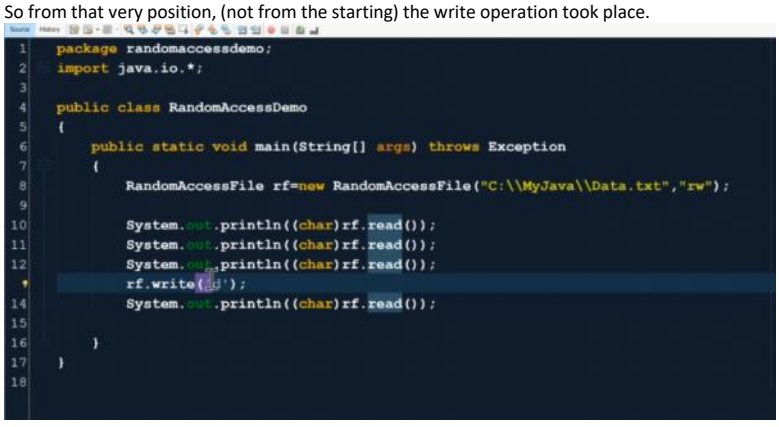


```
package randomaccessdemo;
import java.io.*;

public class RandomAccessDemo
{
    public static void main(String[] args) throws Exception
    {
        RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","rw");

        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        rf.write('d');
    }
}
```



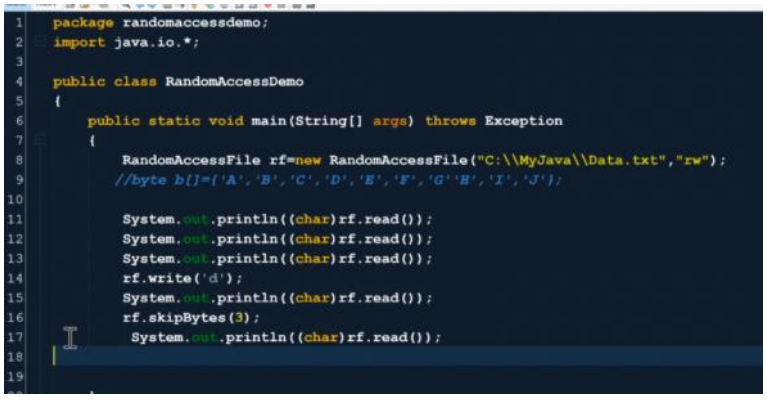
```
C:\MyJava>type Data.txt
ABCDEFGHIJ
C:\MyJava>
```

So from that very position, (not from the starting) the write operation took place.



```
package randomaccessdemo;
import java.io.*;

public class RandomAccessDemo
{
    public static void main(String[] args) throws Exception
    {
        RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","rw");

        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        rf.write('d');
        System.out.println((char)rf.read());

    }
}
```

Now suppose I wanna skip few bytes.



```
package randomaccessdemo;
import java.io.*;

public class RandomAccessDemo
{
    public static void main(String[] args) throws Exception
    {
        RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","rw");
        //byte b[]={'A','B','C','D','E','F','G''H','I','J'};

        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        rf.write('d');
        System.out.println((char)rf.read());
        rf.skipBytes(3);
         System.out.println((char)rf.read());

    }
}
```

```
C
E
I
BUILD SUCCESSFUL (total time: 1 second)
```

```
package randomaccessdemo;
import java.io.*;

public class RandomAccessDemo
{
    public static void main(String[] args) throws Exception
    {
        RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","rw");
        //byte b[]={'A','B','C','D','E','F','G''H','I','J'};

        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        rf.write('d');
        System.out.println((char)rf.read());
        rf.skipBytes(3);
        System.out.println((char)rf.read());
        rf.seek(3);
        System.out.println((char)rf.read());
```

```
E
I
BUILD SUCCESSFUL (total time: 1 second)
```

```
package randomaccessdemo;
import java.io.*;

public class RandomAccessDemo
{
    public static void main(String[] args) throws Exception
    {
        RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","rw")
        //byte b[]={'A','B','C','D','E','F','G''H','I','J'};

        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        rf.write('d');
        System.out.println((char)rf.read());
        rf.skipBytes(3);
        System.out.println((char)rf.read());
        rf.seek(3);
        System.out.println((char)rf.read());
        System.out.println(rf.getFilePointer());
```

o/p -> 4

```
public class RandomAccessDemo
{
    public static void main(String[] args) throws Exception
    {
        RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","rw");
        //byte b[]={'A','B','C','D','E','F','G''H','I','J'};

        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        rf.write('d');
        System.out.println((char)rf.read());
        rf.skipBytes(3);
        System.out.println((char)rf.read());
        rf.seek(3);
        System.out.println((char)rf.read());
        System.out.println(rf.getFilePointer());
        rf.seek(rf.getFilePointer()+2);
```

After Printing,

```
public class RandomAccessDemo
```

```
public class RandomAccessDemo
{
    public static void main(String[] args) throws Exception
    {
        RandomAccessFile rf=new RandomAccessFile("C:\\MyJava\\Data.txt","
        //byte b[]={'A','B','C','D','E','F','G''H','I','J'};

        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        System.out.println((char)rf.read());
        rf.write('d');
        System.out.println((char)rf.read());
        rf.skipBytes(3);
```

```
d
4
BUILD SUCCESSFUL (total time: 1 second)
```

File class

```
package filehandling;
import java.io.*;

public class FileHandling
{
    public static void main(String[] args) throws Exception
    {
        File f=new File("C:\\MyJava");

        System.out.println(f.isDirectory());


    }
}
```

```
package filehandling;
import java.io.*;

public class FileHandling
{
    public static void main(String[] args) throws Exception
    {
        File f=new File("C:\\MyJava");

        System.out.println(f.isDirectory());
        String list[]=f.list();

        for(String x:list)
        {
            System.out.println(x);
        }

    }
}
```

```
Circle.class
Circle.java
Data.txt
Student1.txt
Student2.txt
```

```java
package filehandling;
import java.io.*;

public class FileHandling
{
    public static void main(String[] args) throws Exception
    {
       File f=new File("C:\\MyJava");

       System.out.println(f.isDirectory());
       File list[]=f.listFiles();

       for(File x:list)
       {
           System.out.println(x.getName());
           System.out.println(x.getPath());
       }

    }
}
```

```java
package filehandling;
import java.io.*;

public class FileHandling
{
    public static void main(String[] args) throws Exception
    {
       File f=new File("C:\\MyJava");

       System.out.println(f.isDirectory());
       File list[]=f.listFiles();

       for(File x:list)
       {
           System.out.print(x.getName()+" ");
           System.out.println(x.getPath());
       }

    }
}
```

```
Student4.txt C:\MyJava\Student4.txt
Test C:\MyJava\Test
Test.txt C:\MyJava\Test.txt
BUILD SUCCESSFUL (total time: 1 second)
```

```
Student1.txt C:\MyJava\Student1.txt
Student2.txt C:\MyJava\Student2.txt
Student3.txt C:\MyJava\Student3.txt
Student4.txt C:\MyJava\Student4.txt
Test C:\MyJava\Test
Test.txt C:\MyJava\Test.txt
BUILD SUCCESSFUL (total time: 1 second)
```

```java
package filehandling;
import java.io.*;

public class FileHandling
{
    public static void main(String[] args) throws Exception
    {
        File f=new File("C:\\MyJava\\Data.txt");
        f.setReadOnly();
        FileOutputStream fos =new FileOutputStream("C:\\MyJava\\Data.txt");


    }
}
```

```
compile:
run:
Exception in thread "main" java.io.FileNotFoundException: C:\MyJava\Data.tx
        at java.base/java.io.FileOutputStream.open0(Native Method)
        at java.base/java.io.FileOutputStream.open(FileOutputStream.java:29
        at java.base/java.io.FileOutputStream.<init>(FileOutputStream.java:
        at java.base/java.io.FileOutputStream.<init>(FileOutputStream.java:
        at filehandling.FileHandling.main(FileHandling.java:10)
```

```java
package filehandling;
import java.io.*;

public class FileHandling
{
    public static void main(String[] args) throws Exception
    {
        File f=new File("C:\\MyJava\\Data.txt");
        f.setWritable(true);
        FileOutputStream fos =new FileOutputStream("C:\\MyJava\\Data.txt");
```

```
Deleting: \\Mac\Home\Documents\NetBeansProjects\FileHandling\build\built-ja
deps-jar:
Updating property file: \\Mac\Home\Documents\NetBeansProjects\FileHandling\
Compiling 1 source file to \\Mac\Home\Documents\NetBeansProjects\FileHandli
compile:
run:
BUILD SUCCESSFUL (total time: 1 second)
```

Serialization.

Let's discuss a problem scenario.
Suppose we have a class called Student.

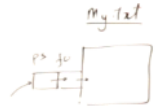
```
class Student
{
    int rollno,
    String name,
    String dept;
}
```

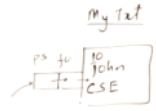
Now I want to store the object of that class in a file.
Suppose file is My.txt

Let's look at the first solution

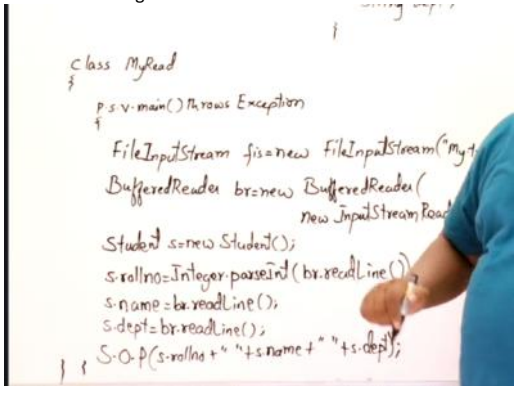
```
class Student
{
    int rollno;
    String name;
    String dept;
}
```

```
class MyWrite
{
  P.S.V.main() throws Exception
  {
    FileOutputStream fo=new FileOutputStream("my.txt");
    PrintStream ps=new PrintStream(fo);
    Student s=new Student();
    S.rollno=10; s.name="John"; s.dept="CSE";
    Ps.println(s.rollno);
    Ps.println(s.name);
    Ps.println(s.dept);
  }
}
```





Now after writing I should be able to read them also.



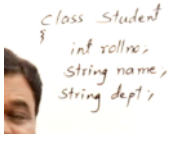
So here in this solution, we can see that object of the Student class is stored by storing one value at a time - Rollno, name, dept - separately.

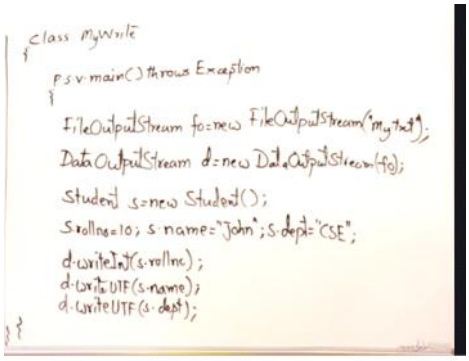
Now in printstream, we can only store string, while rollno. Is integer, so we have to typecast it which is a problem.

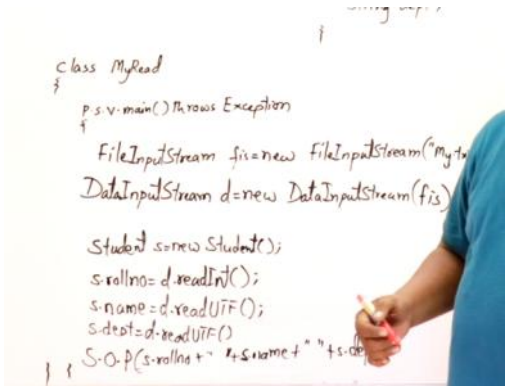
Butt I want data to be stored in its own data type, and read them using same data type.

Serialisation: using DataInput and DataOutput Streams

Consider the previous example only.



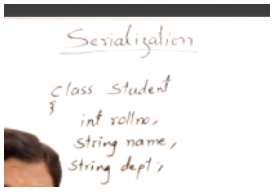


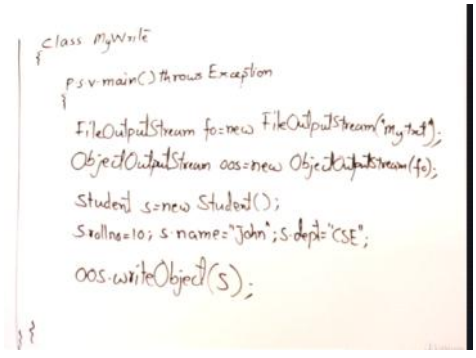


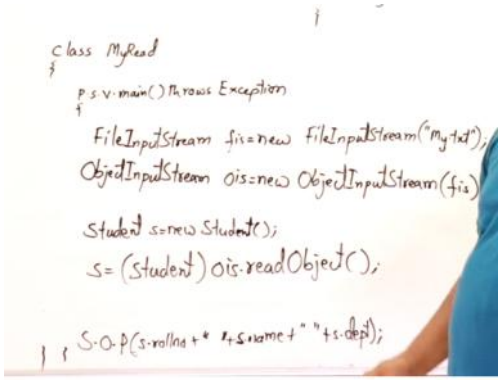
If you writing using data output stream, then u gotta read it using data input stream.

Ok, in this solution we handling the datatypes separately, but despite, we are writing each of the object values separately, How about writing and reading the entire object itself? Welcome to our final solution.

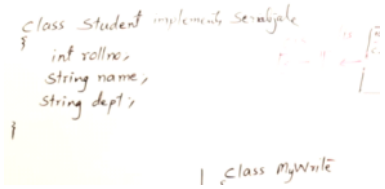
Serialization Final





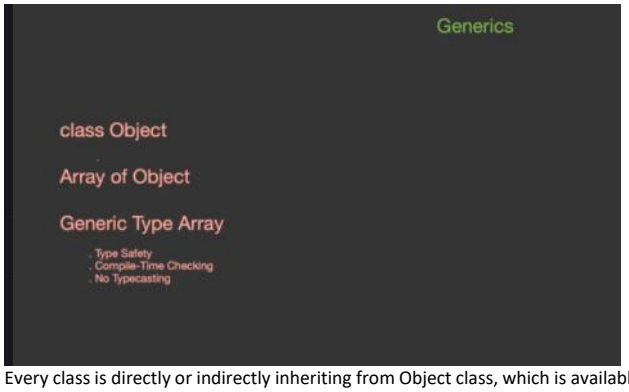


Now implements serializable -> is mandatory

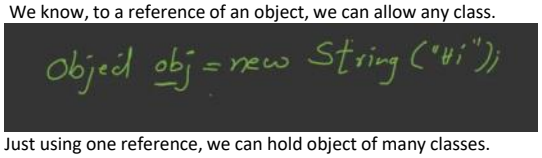


Non parameterized constructor must be there - else reading cannot be done - only writing can be done.
If any static/transient members are there, they won't be serializable.
If any members we don't wanna serialize, make it transient.

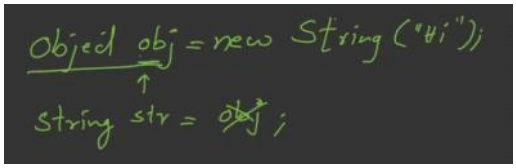
Generics



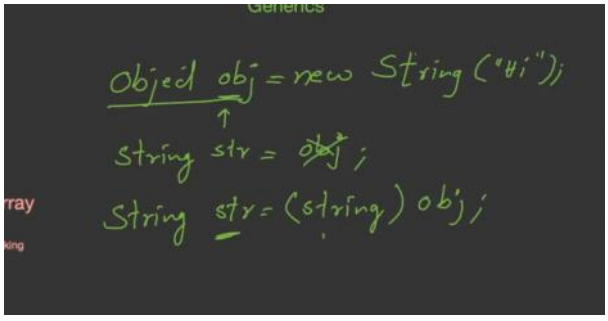
Every class is directly or indirectly inheriting from Object class, which is available in java.lang package.
We know, to a reference of an object, we can allow any class.



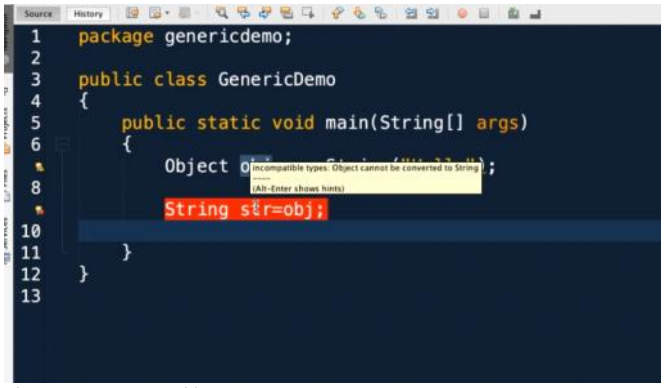
Just using one reference, we can hold object of many classes.
This is nothing but generalization - We can generalise object of any type as an Object
So we achieve generalisation using Object class in java.



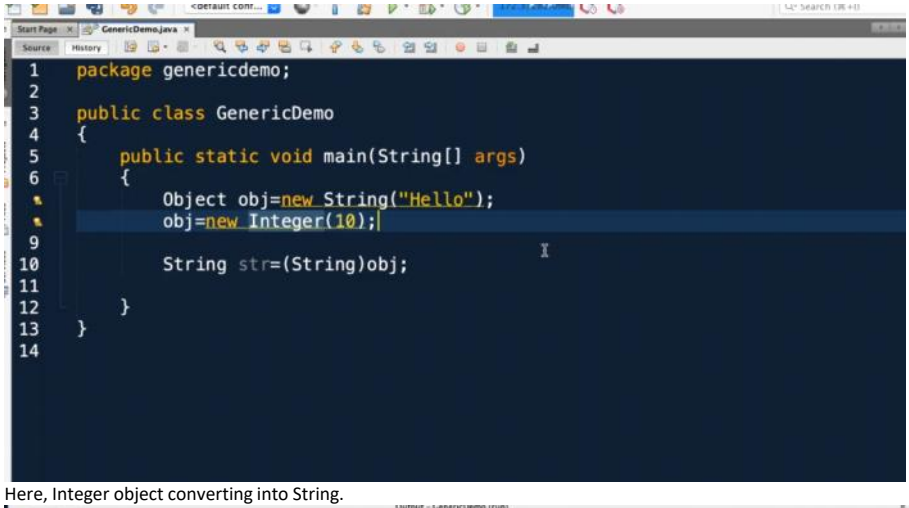
Here 2nd statement is not allowed. Because we can't assign superclass refernce directly - we have to do typecasting.



So, java used Object class vefore to achieve Generalisation.



Above gives incompatible types.



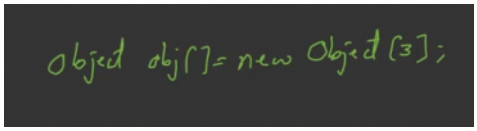
Here, Integer object converting into String.

But we are getting an error.

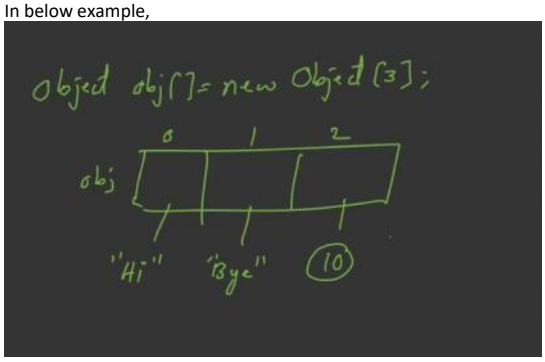
So, if we have reference of an Object, we can assign the object of any type.
When we want to get it back in actual reference, we have to do type-casting.

So, the drawback of using Objects is there is no type-safety.
Here we assigned String also, then integer also to obj.
Another problem is type-casting.
Anpther problem is compiler not checking Im type casting String Type to Integer type as by the above example.

Now, lets see Array of Objects



In below example,



We assigned an integer by mistake in array of objects which was supposed to be string.

```
package genericdemo;

public class GenericDemo
{
    public static void main(String[] args)
    {
        Object obj[]=new Object[3];

        obj[0]="hi";
        obj[1]="bye";
        obj[2]=new Integer(10);

        String str;

        for(int i=0;i<3;i++)
        {
            str=(String)obj[i];
        }
```

```
    {
        Object obj[]=new Object[3];

        obj[0]="hi";
        obj[1]="bye";
        obj[2]=new Integer(10);

        String str;

        for(int i=0;i<3;i++)
        {
            str=(String)obj[i];
            System.out.println(str);
        }

    }
}
```

```
Updating property file: /Users/abdulbari/NetBeansProjects/GenericDemo/build/built-jar.properties
Compiling 1 source file to /Users/abdulbari/NetBeansProjects/GenericDemo/build/classes
compile:
run:
hi
bye
Exception in thread "main" java.lang.ClassCastException: java.lang.Integer cannot be cast to java.
        at genericdemo.GenericDemo.main(GenericDemo.java:17)
/Users/abdulbari/NetBeansProjects/GenericDemo/nbproject/build-impl.xml:1355: The following error o
BUILD FAILED (total time: 0 seconds)
```

10:19 / 16:19

Now lets get ahead with Generics.

Lets consider an array that can store any type of Object. But it should store only Strings.

We can achieve this using generics.

Now willl create an array of type Generics.

T is for generic.

We will take the help of Object and create Generics.

```
package genericdemo;

public class GenericDemo<T>
{
    T data[]=(T[]) new Object[3];


    public static void main(String[] args)
    {

    }
}
```

```
1   package genericdemo;
2
3   public class GenericDemo<T>
4   {
5       T data[]=(T[]) new Object[3];
6
7
8       public static void main(String[] args)
9       {
10              GenericDemo<Integer> gd=new GenericDemo();
11
12      }
13  }
14
```

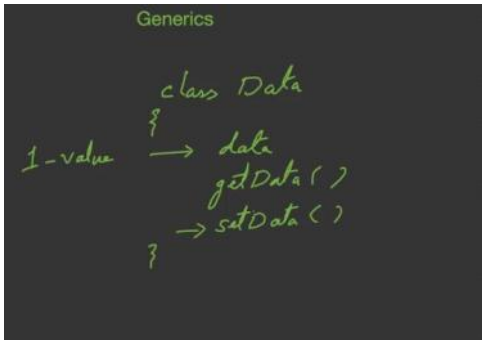
```
1   package genericdemo;
2
3   public class GenericDemo<T>
4   {
5       T data[]=(T[]) new Object[3];
6
7
8       public static void main(String[] args)
9       {
10              GenericDemo<String> gd=new GenericDemo();
11
12
13      }
14  }
15
```

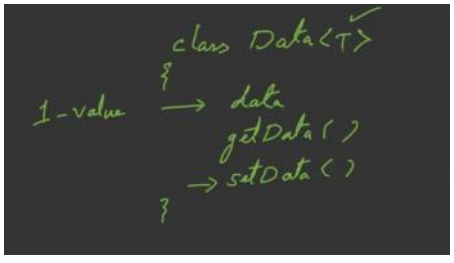
```
1   package genericdemo;
2
3   public class GenericDemo<T>
4   {
5       T data[]=(T[]) new Object[3];
6
7
8       public static void main(String[] args)
9       {
10              GenericDemo<String> gd=new GenericDemo();
11
12              gd.data[0]="hi";
13              gd.data[1]="bye";
14              gd.data[2]=10;
15
16      }
17  }
18
```

Now look it has given error when we used integer!

```java
package genericdemo;

public class GenericDemo<T>
{
    T data[]=(T[]) new Object[3];


    public static void main(String[] args)
    {
            GenericDemo<String> gd=new GenericDemo();

            gd.data[0]="hi";
            gd.data[1]="bye";
           // gd.data[2]=new Integer(10);

           String str=gd.data[0];

    }
```

Look we didn't have to type-cast.






```java
class Data<T>
{
    private T obj;

    public void setData(T v)
    {
        obj=v;
    }
    public T getData()
    {
        return obj;
    }
}
```

```
public class GenericDemo
{
    public static void main(String[] args)
    {
        Data<Integer> d=new Data<>();
        d.setData(10);



    }
}
```

Here autoboxing is happening.

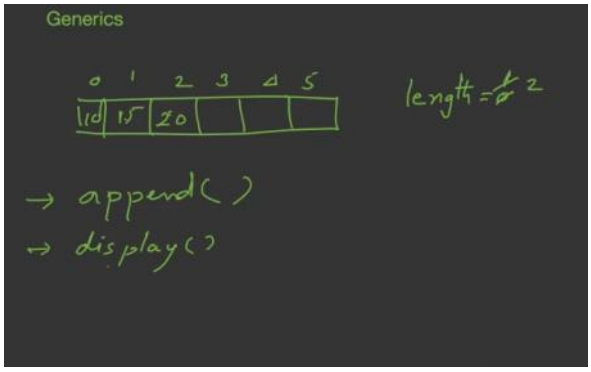
```
public class GenericDemo
{
    public static void main(String[] args)
    {
        Data<Integer> d=new Data<>();
        d.setData(new Integer(10));

        System.out.println(d.getData());

    }
}
```

```
ant -f /Users/abdulbari/NetBeansProjects
init:
Deleting: /Users/abdulbari/NetBeansProje
deps-jar:
Updating property file: /Users/abdulbari
Compiling 1 source file to /Users/abdulb
compile:
run:
10
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
public class GenericDemo
{
    public static void main(String[] args)
    {
        Data<Integer> d=new Data<>();
        d.setData("Hi");
        System.out.println(d.getData());

    }
}
```

Look its givin error for String.

Now lets look at class MyArray



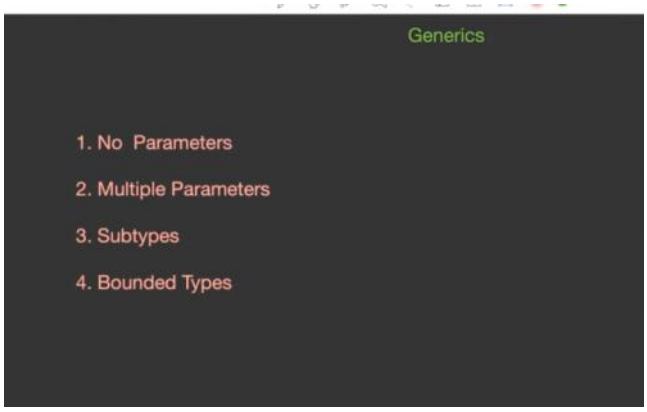
```java
package genericdemo;

class MyArray<T>
{
    T A[]=(T[]) new Object[10];
    int length =0;

    public void append(T v)
    {
        A[length++]=v;
    }

    public void display()
    {
        for(T x:A)
        {
            System.out.println(x);
        }
    }

```

```java
package genericdemo;

class MyArray<T>
{
    T A[]=(T[]) new Object[10];
    int length =0;

    public void append(T v)
    {
        A[length++]=v;
    }

    public void display()
    {
        for(int i=0;i<length;i++)
        {
            System.out.println(A[i]);
        }
    }

```

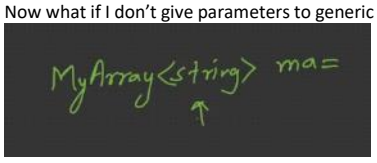
```java
public class GenericDemo
{
    public static void main(String[] args)
    {
        MyArray<Integer> ma=new MyArray<>();

        ma.append(10);
        ma.append(20);
        ma.append(30);

        ma.display();

    }
}
```

```
deps-jar:
Updating property file: /Users/abdulbari/NetB
Compiling 1 source file to /Users/abdulbari/N
Note: /Users/abdulbari/NetBeansProjects/Gener
Note: Recompile with -Xlint:unchecked for det
compile:
run:
10
20
30
BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
package genericdemo;

@SuppressWarnings("unchecked")

class MyArray<T>
{
    T A[]=(T[]) new Object[10];
    int length =0;

    public void append(T v)
    {
        A[length++]=v;
    }

    public void display()
    {
        for(int i=0;i<length;i++)
        {
            System.out.println(A[i]);
        }
```



Now what if I don't give parameters to generic



Now getting ahead with the previous code we done earlier,

```java
public class GenericDemo
{

    public static void main(String[] args)
    {
        MyArray ma=new MyArray();


        ma.append("Hi");
        ma.append("Bye");
        ma.append("Go");

        ma.display();


    }
}
```

So now it will take Object.

```java
public class GenericDemo
{

    public static void main(String[] args)
    {
        MyArray ma=new MyArray();


        ma.append("Hi");
        ma.append(new Integer(10));
        ma.append("Go");

        ma.display();
```

Now it won't restrict.

```
deps-jar:
Updating property file: /Users/abdulbari/NetBeans
Compiling 1 source file to /Users/abdulbari/NetBe
Note: /Users/abdulbari/NetBeansProjects/GenericDe
Note: Recompile with -Xlint:unchecked for details
compile:
run:
Hi
10
Go
BUILD SUCCESSFUL (total time: 0 seconds)
```

We can pass 2 parameters also in generics:

```
package genericdemo;

@SuppressWarnings("unchecked")

class MyArray<T,K>
{
    T A[]=(T[]) new Object[10];
    int length =0;

    public void append(T v)
    {
        A[length++]=v;
    }

    public void display()
    {
        for(int i=0;i<length;i++)
        {
```

```
class MyArray<K,V>
```

```
public class GenericDemo
{

    public static void main(String[] args)
    {
        MyArray<Integer,String> ma=new MyArray();



        ma.display();


    }
}
```

Now lets look at Subtypes

```
package genericdemo;

@SuppressWarnings("unchecked")

class MyArray<T>
{
    T A[]=(T[]) new Object[10];
    int length =0;

    public void append(T v)
    {
        A[length++]=v;
    }

    public void display()
    {
        for(int i=0;i<length;i++)
        {
```

```
class MyArray2 extends MyArray<String>
{

}
```

```
public class GenericDemo
{

    public static void main(String[] args)
    {
        MyArray2 ma=new MyArray2();

        ma.append("HI");
        ma.append("Bye");
        ma.append("Go");


        ma.display();


    }
}
```

Here, MyArray is generic, but MyArray2 is not generic, but will work as String.

```
public class GenericDemo
{

    public static void main(String[] args)
    {
        MyArray2 ma=new MyArray2();

        ma.append(10);
        ma.append("Bye");
        ma.append("Go");


        ma.display();


    }
```

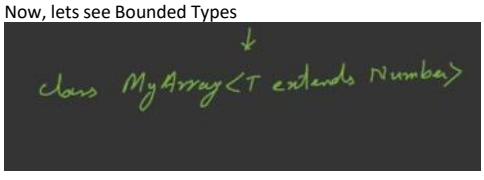
Integer value would be wrong.

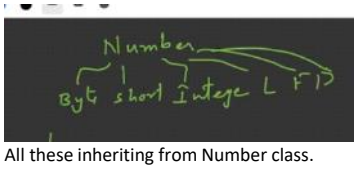
Now if want MyArray2 to act as generic

```
class MyArray2<T> extends MyArray<T>
{

}
```

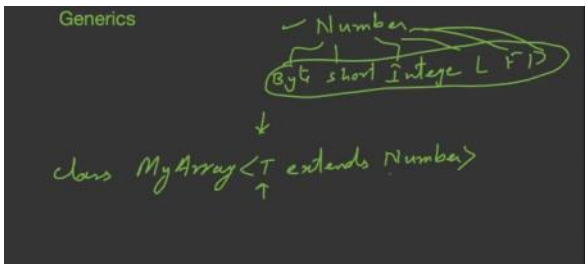
```
public class GenericDemo
{

    public static void main(String[] args)
    {
        MyArray2<String> ma=new MyArray2<>();

        ma.append(10);
        ma.append("Bye");
        ma.append("Go");


        ma.display();
```

Now, lets see Bounded Types





All these inheriting from Number class.




```java
package genericdemo;

@SuppressWarnings("unchecked")

class MyArray<T extends Number>
{
    T A[]=(T[]) new Object[10];
    int length =0;

    public void append(T v)
    {
        A[length++]=v;
    }

    public void display()
    {
        for(int i=0;i<length;i++)
        {
            System.out.println(A[i]);
```

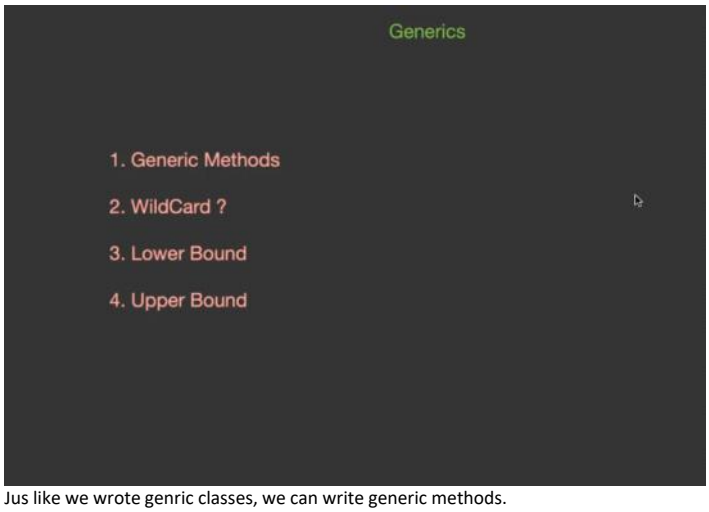
```java
public class GenericDemo
{

    public static void main(String[] args)
    {
        MyArray<Float> ma=new MyArray<>();

        ma.append(12.5f);
        ma.append(6.57f);
        ma.append(9.75f);


        ma.display();

```

```java
package genericdemo;

class A{}
class B extends A{}
class C extends A{}

@SuppressWarnings("unchecked")

class MyArray<T extends A>
{
    T A[]=(T[]) new Object[10];
    int length =0;

    public void append(T v)
    {
        A[length++]=v;
    }

    public void display()
```

```java
ublic class GenericDemo


    public static void main(String[] args)
    {
        MyArray<A> ma=new MyArray<>();



    }
```

```
Source  History
1   package genericdemo;
2
3   interface A{}
4   class B implements A{}
5   class C implements A{}
6
7
8   @SuppressWarnings("unchecked")
9
10  class MyArray<T extends A>
11  {
12      T A[]=(T[]) new Object[10];
13      int length =0;
14
15      public void append(T v)
16      {
17          A[length++]=v;
18      }
```

Here extends works even for interfaces.



Jus like we wrote genric classes, we can write generic methods.

```
9
0   public class GenericDemo
1   {
2       static <E> void show(E[] list)
3       {
4           for(T x)
5       }
6
7       public static void main(String[] args)
8       {
9
0       }
1   }
2
```

```
public class GenericDemo
{
    static <E> void show(E[] list)
    {
        for(E x:list)
        {
            System.out.println(x);
        }
    }

    public static void main(String[] args)
    {

    }
}
```

```java
public class GenericDemo
{
    static <E> void show(E[] list)
    {
        for(E x:list)
        {
            System.out.println(x);
        }
    }

    public static void main(String[] args)
    {
           show(new String[]{"Hi","Go","Bye"});

    }
}
```

```java
public class GenericDemo
{
    static <E> void show(E[] list)
    {
        for(E x:list)
        {
            System.out.println(x);
        }
    }

    public static void main(String[] args)
    {
           show(new String[]{"Hi","Go","Bye"});
           show(new Integer[]{10,20,30,40});
    }
}
```

```
Updating property file: /Users/abdulbari/NetBe
Compiling 1 source file to /Users/abdulbari/Ne
compile:
run:
Hi
Go
Bye
10
20
30
40
BUILD SUCCESSFUL (total time: 0 seconds)
```

2x 3:45 / 11:37

Overview Q&A Notes Announcemen

```java
public class GenericDemo
{
    static <E> void show(E... list)
    {
        for(E x:list)
        {
            System.out.println(x);
        }
    }

    public static void main(String[] args)
    {
           show("Hi","Go","Bye");
           show(10,20,30,40);
    }
}
```

Now lets see Bound type works or not.

```
public class GenericDemo
{
    static <E extends Number> void show(E... list)
    {
        for(E x:list)
        {
            System.out.println(x);
        }
    }

    public static void ma
    {
        show("Hi","Go","Bye");
        show(10,20,30,40);
    }
}
```

method show in class GenericDemo cannot be applied to given types;
required: E[]
found: String,String,String
reason: inference variable E has incompatible bounds
lower bounds: Number
lower bounds: String
where E is a type-variable:
E extends Number declared in method <E>show(E...)
(Alt-Enter shows hints)

Now lets see Wildcards.
Now we will make parameters as generic.

```
public class GenericDemo
{
    static void fun(MyArray obj)
    {
        obj.display();
    }

    public static void main(String[] args)
    {
        MyArray<String> ma1=new MyArray<>();
        ma1.append("Hi");
        ma1.append("Bye");

        MyArray<Integer> ma2=new MyArray<>();
        ma2.append(10);
        ma2.append(20);
    }
}
```

```
30  public class GenericDemo
31  {
32      static void fun(MyArray obj)
33      {
34          obj.display();
35      }
36
37      public static void main(String[] args)
38      {
39          MyArray<String> ma1=new MyArray<>();
40          ma1.append("Hi");
41          ma1.append("Bye");
42
43          MyArray<Integer> ma2=new MyArray<>();
44          ma2.append(10);
45          ma2.append(20);
46
47          fun(ma1);
48          fun(ma2);
49
```

```
init:
Deleting: /Users/abdulbari
deps-jar:
Updating property file: /U
Compiling 1 source file to
compile:
run:
Hi
Bye
10
20
BUILD SUCCESSFUL (total ti
```

Output | Notifications | Finished building GenericD

But we can't use generic type <T> in argument

```
{
    static void fun(MyArray<T> obj)
    {
        obj.display();
    }
```

```java
public class GenericDemo
{
    static void fun(MyArray<?> obj)
    {
        obj.display();
    }

    public static void main(String[] args)
    {
        MyArray<String> ma1=new MyArray<>();
        ma1.append("Hi");
        ma1.append("Bye");

        MyArray<Integer> ma2=new MyArray<>();
        ma2.append(10);
        ma2.append(20);

        fun(ma1);
        fun(ma2);
```

Now '?' is wildcard.
Above is unbounded wildcard. So MyArray obj is same as MyArray<?> obj.

```java
public class GenericDemo
{
    static void fun(MyArray<? extends Number> obj)
    {
        obj.display();
    }

    public static void main(String[] args)
    {
        MyArray<String> ma1=new MyArray<>();
        ma1.append("Hi");
        ma1.append("Bye");

        MyArray<Integer> ma2=new MyArray<>();
        ma2.append(10);
        ma2.append(20);

        fun(ma1);
        fun(ma2);
```

Now this won't access String nomore.

```java
public class GenericDemo
{
    static void fun(MyArray<? super Number> obj)
    {
        obj.display();
    }

    public static void main(String[] args)
    {
        MyArray<String> ma1=new MyArray<>();
        ma1.append("Hi");
        ma1.append("Bye");

        MyArray<Integer> ma2=new MyArray<>();
        ma2.append(10);
        ma2.append(20);

        fun(ma1);
        fun(ma2);
```

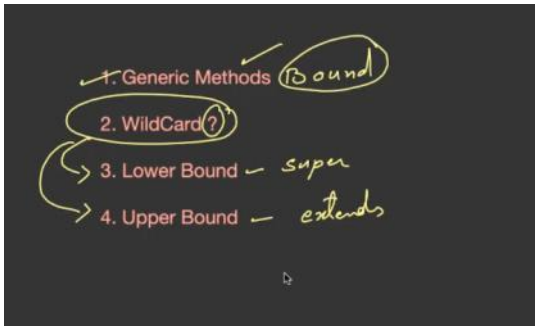
We re allowing those classes which are superclasses of Number.
Lets see another example

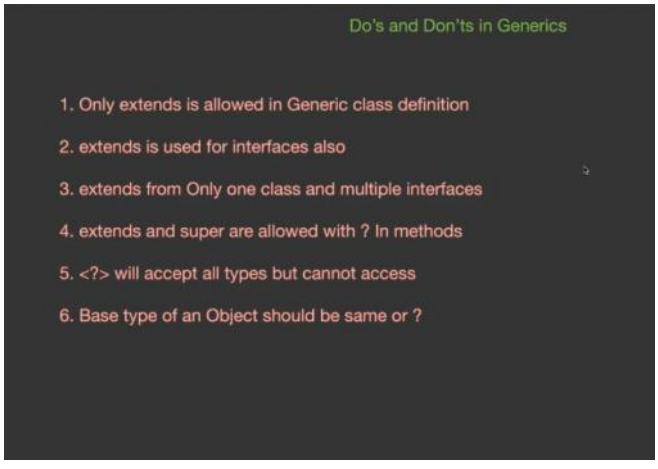
```java
package genericdemo;

class A{}
class B extends A{}
class C extends B{}
```

```java
public class GenericDemo
{
    static void fun(MyArray<? extends A> obj)
    {
        obj.display();
    }

    public static void main(String[] args)
    {
        MyArray<B> ma1=new MyArray<>();


        MyArray<C> ma2=new MyArray<>();
```

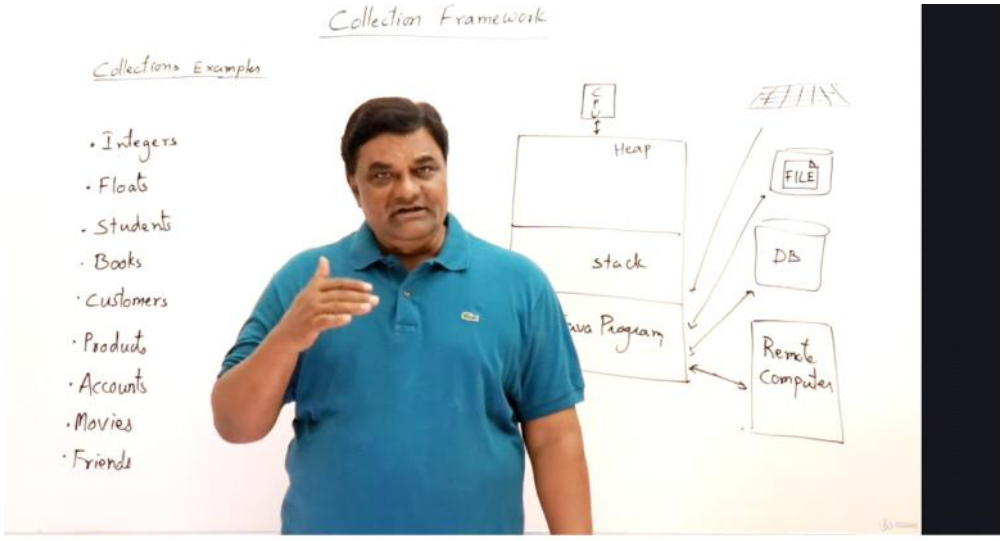
```java
public class GenericDemo
{
    static void fun(MyArray<? extends A> obj)
    {
        obj.display();
    }

    public static void main(String[] args)
    {
        MyArray<B> ma1=new MyArray<>();


        MyArray<C> ma2=new MyArray<>();

        fun(ma1);
        fun(ma2);
```

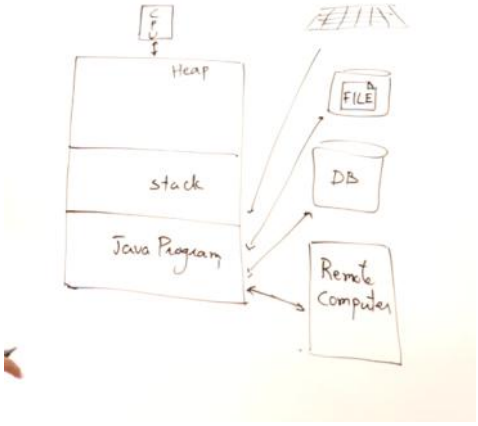
```java
ublic class GenericDemo

    static void fun(MyArray<? super B> obj)
    {
        obj.display();
    }

    public static void main(String[] args)
    {
        MyArray<B> ma1=new MyArray<>();


        MyArray<C> ma2=new MyArray<>();

        fun(ma1);
        fun(ma2);
```

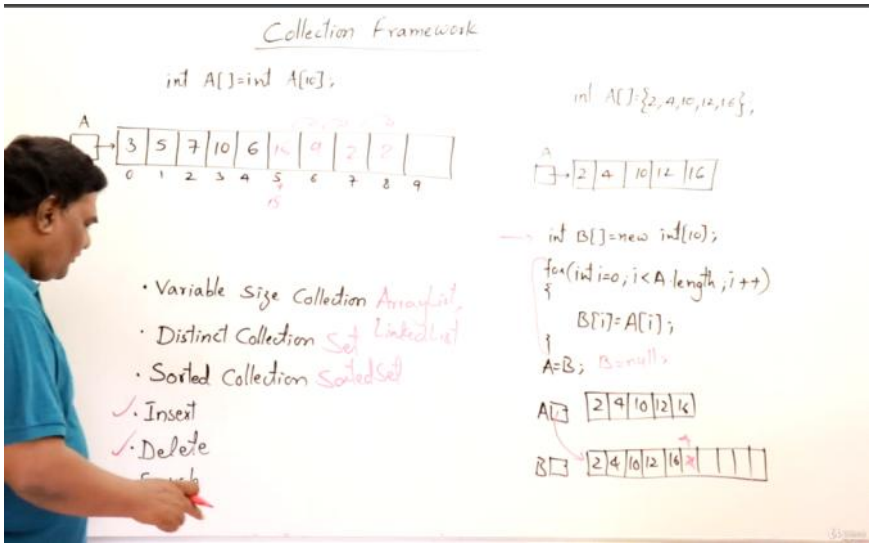


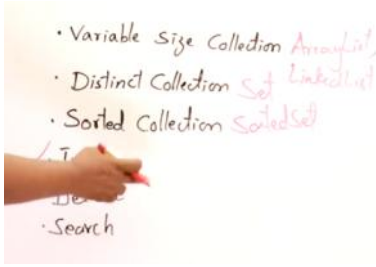


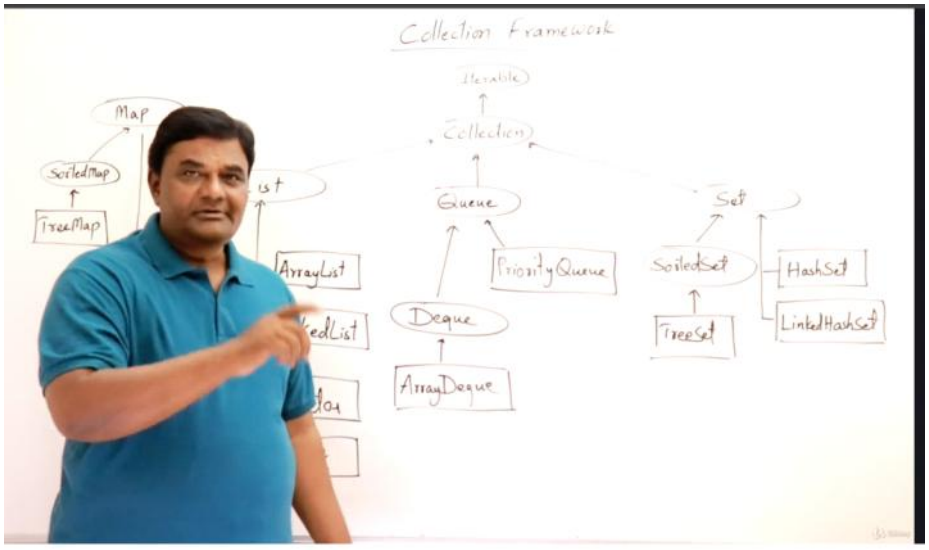
Collections

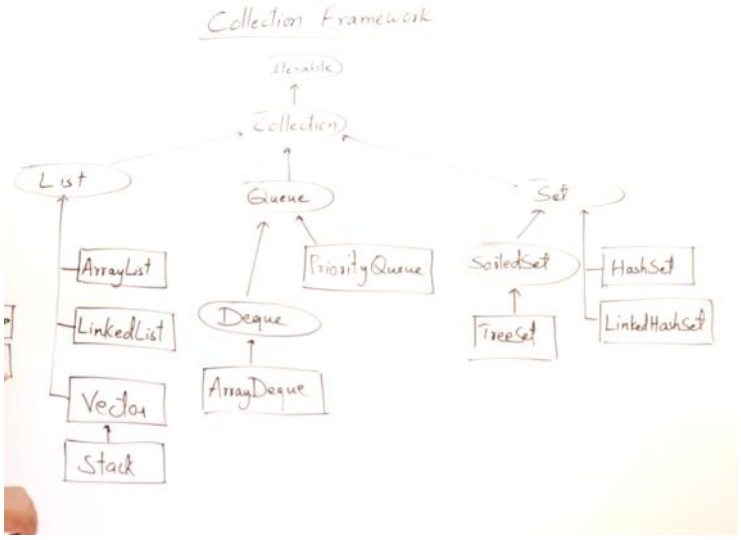

Collection Framework
Collections Examples
• Integers
• Floats
• Students
• Books
• Customers
• Products
• Accounts
• Movies
• Friends
Heap
stack
Java Program
FILE
DB
Remote Computer

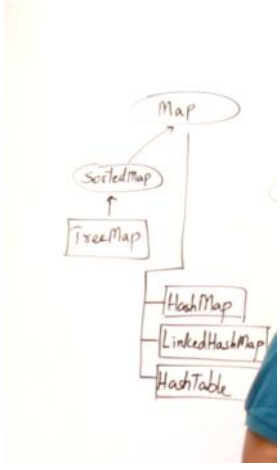

Heap
stack
Java Program
FILE
DB
Remote Computer


Collection Framework
int A[]=int A[10];
int A[]={2,4,10,12,16};
int B[]=new int[10];
for(int i=0;i<A.length;i++)
B[i]=A[i];
A=B;
• Variable Size Collection ArrayList,
• Distinct Collection Set LinkedList
• Sorted Collection SortedSet
• Insert
• Delete


• Variable Size Collection ArrayList,
• Distinct Collection Set LinkedList
• Sorted Collection SortedSet
• Search

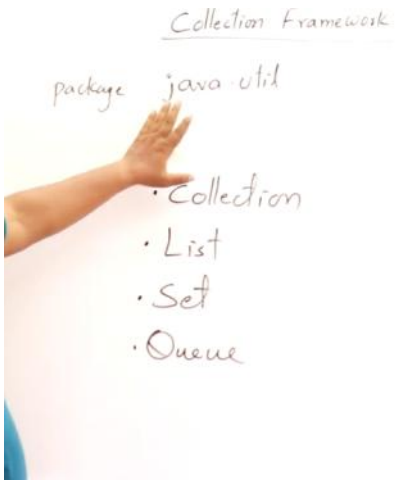


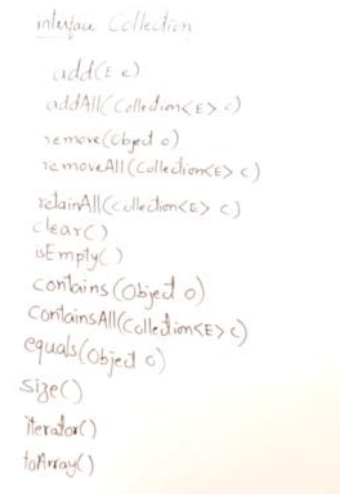




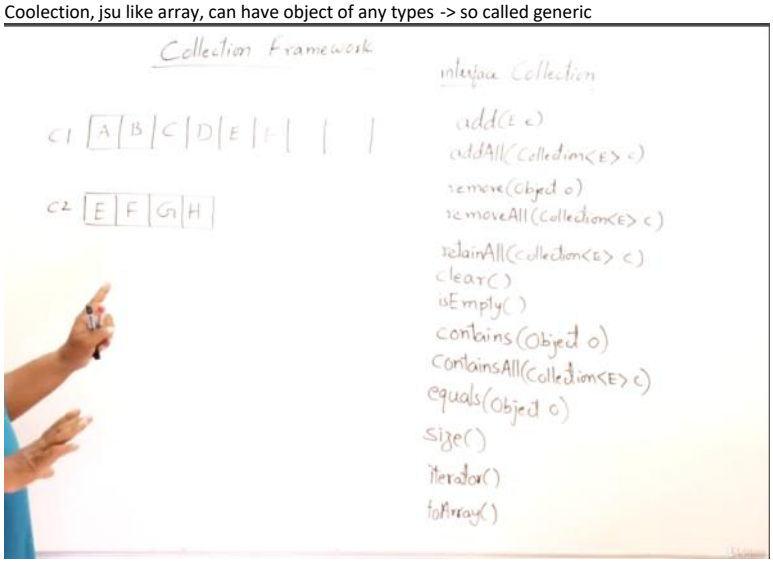
Things in oval shape are interfaces, and things in rectangular shapes are classes.

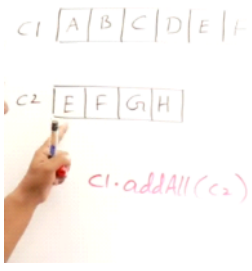
Understanding Collection interface

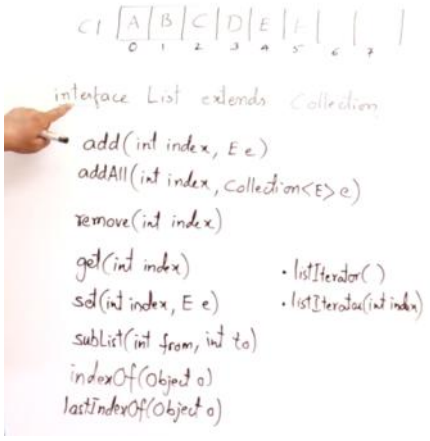




Coolection, jsu like array, can have object of any types -> so called generic






c1 [A][B][C][D][E][ ][ ][ ]
0 1 2 3 4 5 6 7
interface List extends Collection
add(int index, E e)
addAll(int index, Collection<E> e)
remove(int index)
get(int index)
set(int index, E e)
subList(int from, int to)
indexOf(Object o)
lastIndexOf(Object o)
• listIterator( )
• listIterator(int index)