Week 4

15 July 2023 03:03 PM

class date- 15th July 2023
String class-

1) String is a final class.
2) We can't create or make child class of String class.
3) String class is present in java.lang package.
4) There is no need to import java . lang package.
5) String class is Immutable.(Non-Changeable)
6) We can create Object for String class in 3 ways.
a) Using new keyword. b) Using String Literal. c) Due to Runtime Operation.

1) String class Object are Immutable. (Non-Changeble)	1) StringBuffer class Object are muatble in nature.

Ex
String s=new String ("ABC");
s.concat("PQR");
System.out.println(s); // ABC
	StringBuffer sb=new StringBuffer("ABC");
	sb.append("PQR");
	System.out.println(sb); // ABCPQR

Note : If create String class Object , then we cant change or modify its Content.	Note : If we create StringBuffer Class Object then we can modify ior change its content.

Purpose String class equals method() ?

==========

=> Object class equals method is used for referece Comparaison

2) equals method of Object class is Override in String class.

=> String class equals method is used for content comparison.

```
String   s1= new String ("ABC");
String   s2=new String("PQR");
String   s3=new String("ABC");
boolean b=s1.equals(s2);
System.out.pritnln(b);  //false
```

Object creation For String Class:

==========

Using new Keyword:

==========

Object creation in String:

==========

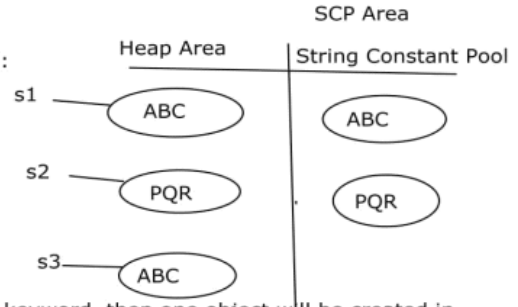
1) we can create Object of String class in 3 ways.
   a) Using new Keyword
   b) Using String Literal
   c)Using Runtime Operation.

a) String class Object creation Using new keyword:

```
String s1=new String("ABC");

String s2=new String("PQR");

String s3=new String("ABC");
```



=> Whenever we create Object of String using new keyword, then one object will be created in Heap area and one Object will be created in SCP area.(for future Purpose).
=> IN Heap area compulsory object will be created.(Duplicate Object can store in Heap Area)
=> Object creation ini SCP area is Optional process.(Duplicate Object can not create in SCP area)

Using String Literal :

==========

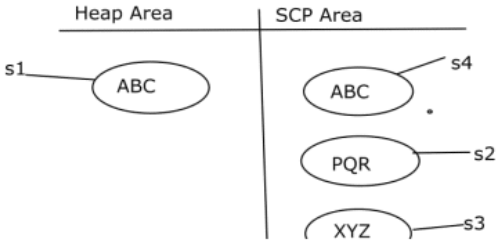
String Object creation Using String Literals:

Ex . String s1= new Sring ("ABC");

String s2="PQR";

String s3="XYZ";

String s4="ABC";

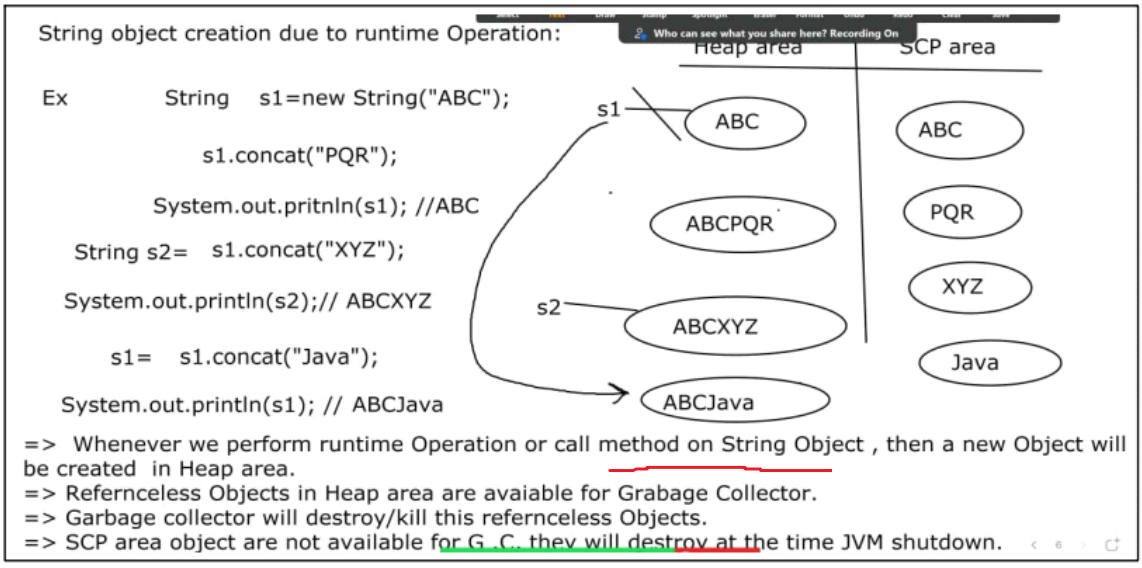


String s3="XYZ";

String s4="ABC";

=>Whenever we create Object of String class Using String Literal, then Object will be created in only in SCP area.
=>If same content Object is already available in SCP area ,then JVM will does not create new Object.
=> Same existing object will be resued.

Using Runtime Operation:
==========



In the above example,
In s1.concat("PQR"); we are passing the string literal, so ABC will be created in SCP area. Then because of **s1.** , the content of s1 which is ABC and PQR become ABCPQR which gets created in heap memory. s1.concat("PQR") is reference less object.

s2= s1.concat("XYZ");
s1 is pointing to ABC. Because of literal, XYZ will be created in SCP area and because of **s1. ,** ABCXYZ will be created in heap memory and s2 will point to it.

String s1= "JAVA";
String s2=s1.toLowerCase();
This will create an object in heap memory only and s2 will point to it.

Notes by sir-

Important Methods of String class :
//concat() => to join Strings
//equals() => used for Content checking
//equalsIgnoreCase() => used for Content checking and Ignore the Case
//toLowercase() => to convert content in lowercase
//toUpperCase() => to convert content in Uppercase
//length() => to find out charcter present in String
//indexOf() => find the index position of given character form the start
//lastIndexOf() => find the index position of given character form the End
//substring(int start) => return the substring from the given index
//subString(int st , int end) => return the substring between the given start index and End index
//charAt(int index) => return the character from the Specified index Position
//trim() => removes the white spaces from start and ending of Given String
//compareTo()=> compare the Strings Lexicographically

==========

```
/* compareTo() => Method
* String s1="AAA"; //65
*
* String s2="aaa"; //97
*
* System.out.println(s1.compareTo(s2));

 equalsIgnoreCase() => method
 String s1="java";
 String s2="Python";
```

```
 String s3="JAVA";
 System.out.println(s1.equalsIgnoreCase(s2)); //false
 System.out.println(s1.equalsIgnoreCase(s3)); //true
============================================================
 trim() ==> Method
 String s1=" Complete Java Classes ";
System.out.println(s1);
 String s2=s1.trim(); System.out.println(s2); //

/*
* String s1="Java"; // J a v a
* // 0 1 2 3
* char ch=s1.charAt(0);
* System.out.println(ch); //v
*/ ============================================================
/*
* String s1="Complete java Classes"; //C o m p l e t e J a v a Classes
* // 0 1 2 3 4 5 6 7 8 9
*
* String s2=s1.substring(9);
* System.out.println(s2); //java Classes
*
* String s3 = s1.substring(9, 13);
* System.out.println(s3);//Java
*/ ============================================================
/* indexof()
* lastindexOf()
* String s1="Java"; // J a v a
* // 0 1 2 3
*
* int index=s1.indexOf('a');
* System.out.println(index); //1
*
* int ind= s1.lastIndexOf('a');
* System.out.println(ind); // 3
*
*/ ============================================================
/* length() => Method
* String s1="Complete java Classes";
*
* int len=s1.length();
*
* System.out.println(len); //
*/ ============================================================
/* toUpperCase() => method
* String s1="virat Kohli";
*
* String s2=s1.toUpperCase();
*
* System.out.println(s2); //VIRAT KOHLI
*/ ============================================================

New Section 22 Page 2

/* toLowerCase() => method
* String s1="COMPLETEJAVACLASSES";
*
* String s2=s1.toLowerCase();
*
* System.out.println(s2); //completejavaclasses
*/ ============================================================
/* equals() => Method
* String s1=new String("ABC");
*
* String s2=new String("PQR");
*
* String s3=new String("ABC");
*
* System.out.println(s1.equals(s2)); //false
```

```
* System.out.println(s1.equals(s3)); //true
*/ ============================================================
/* concat() => method
* String s1=new String("Complete");
*
* String s2=s1.concat("JavaClassses");
*
* System.out.println(s2);//CompleteJavaClasses
*/
=============================================================================================
```

String s= new String("ABC");
On printing s, s won't give a memory address because toString method is there in String class.

StringBuffer sbf= new StringBuffer("ABC");
String s1= new String("ABC");
System.out.println(sbf.equals(s1)); //**false**

String s2= new String("XYZ");
s2= new String("456");

String s2= new String("123");
String s3= "123";

```
System.out.println(s2==s3);//false
System.out.println(s2.equals(s3));//true
```

```
String s1= new String("ABC"); //A= 65
String s2= new String("abc"); //a= 97
System.out.println(s1.compareTo(s2)); //65-97= -32
System.out.println(s2.compareTo(s1)); //97-65= 32
System.out.println(s1.compareTo(s1)); //0
```

---

class Date- 16th July 2023
Topics covered: Marker interface, abstract class, Encapsulation, Exception handling- try-catch
I'm saving the images of notebook pages and they're in PDF.

16/07/23

Marker interface = Tag interface

public interface I { } — No variables & methods
- used to make group, tag, to mark

eg Serializable, Clonable, Remut

Q.1) What is marker or tag interface?

→ interface which has blank body, (i.e no methods & no variables)

Q.2) What is the use of marker interface?

→ marker interface is used for decision making purpose.

Q.3) Which are the marker interfaces given by Java?

→ Serializable, Clonable, Remut

abstract class

① abstract class means partial class

② If class has any one method abstract, then class must be abstract.

③ abstract class has both types of method - implemented as well as abstract

```
public abstract class A
{
public void m1(){
}
abstract void m2();
}
```

~~public abstract class A~~
~~{~~
~~public void m1()~~

④ If class has all implemented methods, then also we can make the class as abstract

```
public abstract class A
{
    public void m1(){
    }

    public void m2(){
    }
}
```

Q.4> Why do we make class as abstract if class can have all implemented methods?

→ We want to restrict a class, so that no one can create object of abstract class.

```
X A a = new A();
```

Q.5> How can we call abstract class's implemented methods?

→ By using its child class, we can call abstract class's implemented methods.

~~for~~ for a class, both final & abstract are not possible.

Q.6> How can we create child class of abstract class which has both types of methods (implemented & abstract)?

→ Option (i) child class must implement all abstract methods of abstract class

Option (ii) If child class is unable to implement all abstract methods of abstract class, then make that child class as abstract.

Q.7) Is there constructor inside abstract class?

→ Yes. ~~What is the use~~

Q.8) What is the use abstract class constructor?

→ Abstract class's constructor is used for initializing instance variable.

Q.9) How will abstract class's constructor be called?

→ child class's constructor's first line - super(),
using that, abstract class's constructor will be called.

Encapsulation

Using encapsulation → data hiding

① Wrapping of method and variable in class is called as encapsulation.

② Every java class is example of encapsulation. Java provides 4 access modifiers & with the help of those modifiers we achieve encapsulation. public, protected, default, private.
~~public, prot~~

- public - can be accessed from anywhere
- protected - can be accessed within package & outside the package through subclass
- default - It will be accessed within a package only.
- private - It will be accessed within a class.

getter, setter class is example of proper encapsulation

③ Data hiding is achieved through encapsulation.

```
public class A                    C extends A.
{ protected int x;                { A a = new A();
}                                   method(){
                                    //error
                                    a.x x x
                                  C c1 = new C();
```

protected variable – in child class A outside the package only child's reference & child's object & via child's object, protected variable can be used.

```
package com.cjc.karvenagar;
public class A
{
    protected int x;
}
```

```
package com.cjc.karvenagar
public class B
{
    A a = new A();
    public void m1()
    {
        a.x++;
    }
}
```

```
package com.cjc.akkurdi;
import com.cjc.karvenagar;
public class C extends A
{
    A a = new A();
    public void m4()
    {
        // error - System.out.println(a.x);
        C c = new C();
        System.out.println(c.x);
        Sysout.(x);
    }
}
```

```
package com.cjc.akkurdi;
public class D extends C
{
    public void m5()
    {
        C c = new C();
        // error - Sysout(c.x);
        D d = new D();
        System.out.println(d.x);
        System.out.println(x);
    }
}
```

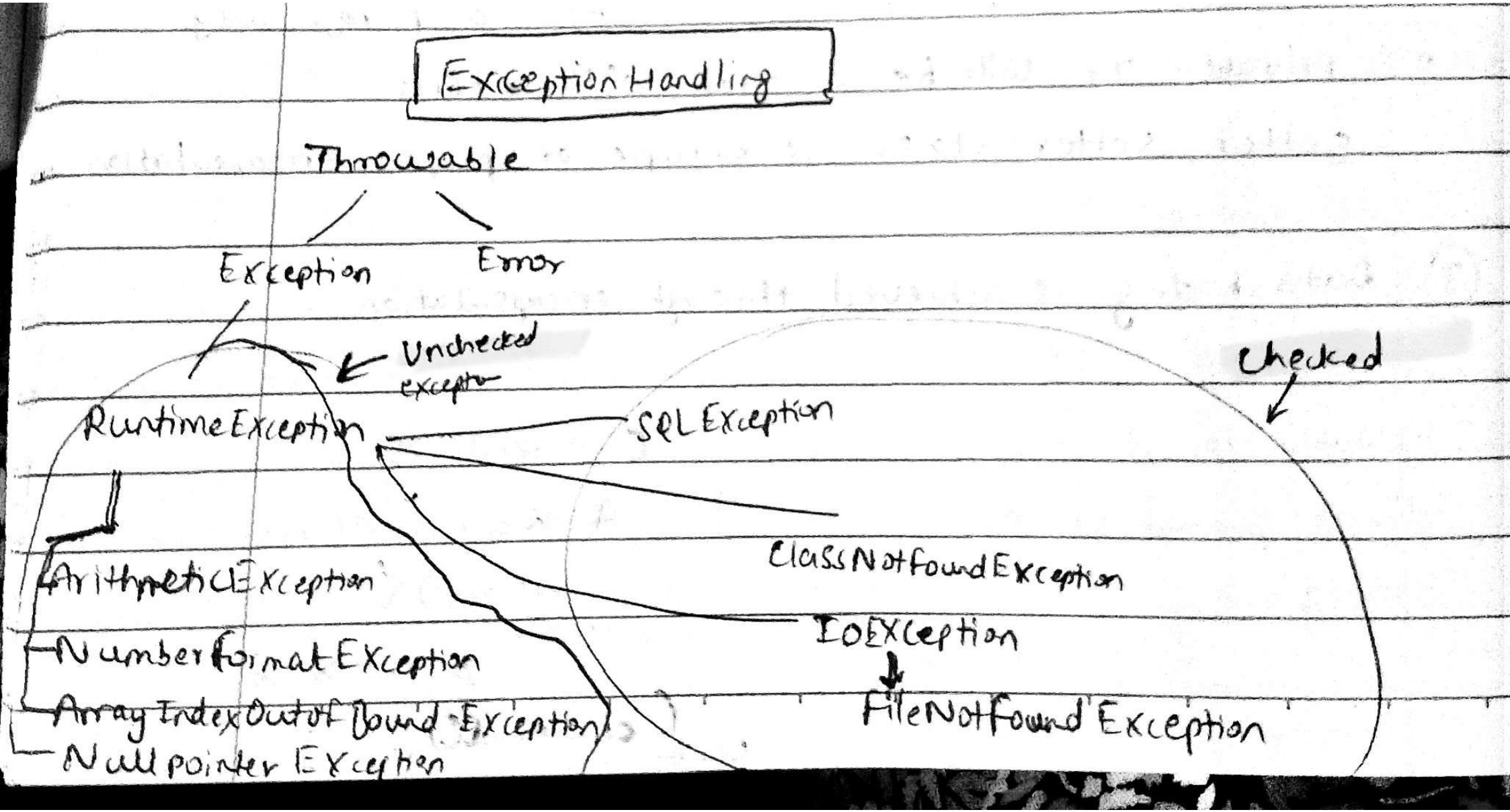
```
package com.cjc.akurdi;
import com.cjc.karvenagar;
public class C extends A
{
  A a = new A();
  public void m4()
  {
// error - System.out.println(a.x);
  C c = new C();
  System.out.println(c.x);
  sysout.(x);
  }
}
```

```
package com.cjc.karvenagar;
public class A
{
  protected int x;
}
```

```
package com.cjc.karvenagar
public class B
{
  A a = new A();
  public void m1()
  {
    a.x++;
  }
}
```

```
package com.cjc.akurdi;
public class D extends C
{
  public void m5(){
    C c = new C();
    // error - Sysout(c.x);
    D d = new D();
    System.out.println(d.x);
  } System.out.println(x);
}
```

Exception Handling



In our program, there're chances of occurence of unwanted situation or abnormal condition which will disturb our normal flow of program. Sometimes our program will get terminated abnormally. For avoiding this problem, we'll use Exception handling.

Exception - unwanted situation or abnormal condition which will disturb our program.

Every exception occurs during running of program.

There're several reasons because of which Exception occurs in our program -

1. Wrong input provided by end user
2. Sometimes we developer make logical mistakes
3. File is misplaced from its existing location.

~~Error class~~

Every error occurs during running of program. Error needs not to be handled. It needs to be solved.

Error occurs because of resource shortage.

Exception handling keywords →

→ try, catch, finally, throw, throws

```
public class Test
{
    p.s.v. main (String [] args)
    {
        sysout. ("main -- start");
        try {
            int x = 10/0;
        }
        catch (ArithmeticException e)
        {
            sysout. ("catch -- block");
        }
        sysout. ("main -- end");
}}
```

Output :-

main -- start

catch -- block

main -- end

Code that may generate problem should be written in try block.
Solution code should be written in catch block.
If there is no problem in try block, then catch block code will be skipped.
If problem occurs inside try block, then remaining lines of code in try-block will not be executed.

```
public class Test
{
    p.s.v. main(String[] args)
    {   try {
            System.out.println("try--start");
            int x = 10/0;
            System.out.println("try--end");
        }
        catch(ArithmeticException e)
        {   System.out.println("catch--block");
        }
        System.out.println("main--end");
    }
}
```

Output:-

```
try--start
catch--block
main--end
```

```
public class Test
{   p.s.v. main(String[] args)
    {   System.out.println("main--start");
        try{
            System.out.println("try--start");
            int x = 10/2;
            System.out.println("try--end");
        }
```

```
catch {
        System.out.println("catch--block");
}

System.out.println("main--end");
}
}
```

Correction- catch(ArithmeticException e)

Output :- main--start

try--start

try--end

main--end