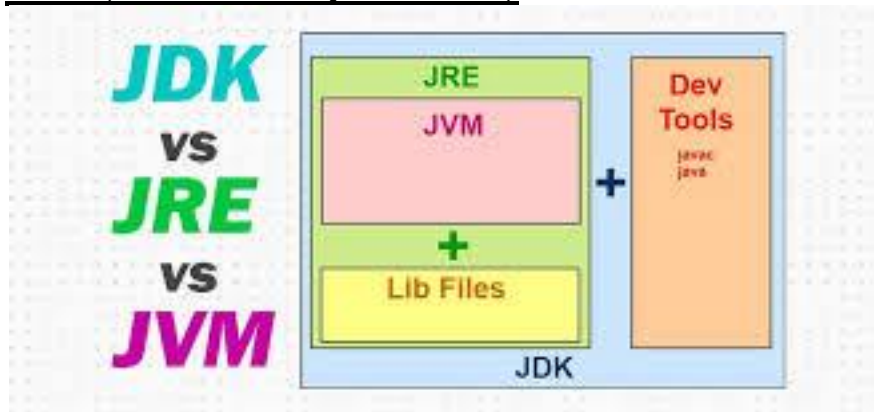


21/02/2019

JDK (Java development kit)



JDK = JRE(Java run time environment)+ Developmental tools

JRE = JVM(Java virtual machine)+Library files

- Developmental tools includes javac, java etc. Javac acts as the compiler that converts high level language to machine language.
- Compiled file is sent to JVM.

Installation and configuration

JDK 8u201 [checksum](#)
JDK 8u202 [checksum](#)

Java SE Development Kit 8u201		
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.		
<input checked="" type="radio"/> Accept License Agreement <input type="radio"/> Decline License Agreement		
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.98 MB	jdk-8u201-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.92 MB	jdk-8u201-linux-arm64-vfp-hflt.tar.gz
Linux x86	170.98 MB	jdk-8u201-linux-i586.rpm
Linux x86	185.77 MB	jdk-8u201-linux-i586.tar.gz
Linux x64	168.05 MB	jdk-8u201-linux-x64.rpm
Linux x64	182.93 MB	jdk-8u201-linux-x64.tar.gz
Mac OS X x64	245.92 MB	jdk-8u201-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	125.33 MB	jdk-8u201-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	88.31 MB	jdk-8u201-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	133.99 MB	jdk-8u201-solaris-x64.tar.Z
Solaris x64	92.46 MB	jdk-8u201-solaris-x64.tar.gz
Windows x86	197.66 MB	jdk-8u201-windows-i586.exe
Windows x64	207.46 MB	jdk-8u201-windows-x64.exe

- Windowsx64 is to be used for 64 bit PC. Once installed the kit needs to be configured with the PC using the steps below.
- My computer → Right click → properties → Advanced system settings
- Advanced s/m settings → Environment variables → New →
- New → Variable name :PATH
Variable value:C folder –pgm files-jdk-bin-copy the address and paste as variable value

Sample program

```

class Sample{
    public static void main(String arg[])
    {
        System.out.print("Hello java");
    }
}

```

↗ **Class name**

- Save the file as Sample.java, where **Sample** is the class name
- Always ensure that **file name is same as the class name** while the file is saved.
- C:\Users\user>g:
- G:\>**cd javapg**
- G:\javapg>**javac Sample.java** → Compilation
- G:\javapg>**java Sample** → Run
- G:\javapg>**cd..** → To go back to the folder

Here G is the drive with the folder whose folder name is javapg inside which the pgm Sample is saved.

Errors

- 2 types: **compilation time error & run time error**
- Compilation time error: deals with syntax mistakes
- Run time error:deals with logical mistakes eg: division by zero

Data types

- 2 types: **Primitive & Non primitive**
- Primitive- int , long, short, byte, char, boolean, float, double

- Non primitive-String, Array, Integer
- Non primitive data types are also known as **wrapper classes or predefined classes**



Variable

- **3 types-local, instance, static**
- **Local**- declared inside method
- **Instance**-declared outside method and inside class
 - It can be used as local variable

e 10

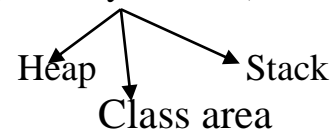
e 20

*The value 10 and 20 gets stored in two different locations in **heap or stack**

*It cannot be used as local variable.

- **Static**- declared with static key word, eg: static int f
 - * It can be used as a local variable
 - * It saves memory
 - *It gets memory only once in **class area**(memory in JVM)

e 20



The value 20 gets stored after erasing the value 10 that was previously stored in e .

- **println** ensures that the next output is printed on the next line.

22/02/2019

Conditional Statements

1. Simple If

if(condition)

```
{  
    TRUE  
}  
FALSE
```

2. If Else

```
if (condition)  
{  
  
    TRUE  
}  
Else  
{  
    FALSE  
}
```

3. If else if

```
if(condition1)  
{  
  
}  
else if(condition 2)  
{  
  
}  
else if(condition n)  
{  
  
}  
}
```

Loops

1.for

```
for(declaration/initialisation;condition;increment/decrement)  
{  
  
}
```

2.While

```
while(condition)
{

}
```

- It is entry controlled

3. Do While

```
do
{

}
while(condition);
```

- It is exit controlled
- It runs the loop once without checking the condition due to the usage of “do” initially

4. Switch

```
switch(i)

{ case 1:{
        break;
    }
  case 2:{
        break;
    }
  case n:{
        break;
    }
  default:{
        break;
    }
}
```

- 1,2..n are integers hence “i” is a variable of int data type and needs to be initialised before using it .

Continue and break statements

- **continue** is used to skip the remaining statements in the loop and to continue with the next iteration.
- **break** statement stops the iteration once and for all and hence no more execution occurs.

25/02/2019

Java Comments

- **Single Line**- is turned into a comment using double slash(//) at the beginning.
Eg: `//int i;`
`//a=b+c;`
- **Multi Line**- is turned into a comment using `/*` at the beginning and `*/` at the end
Eg: `/* int i;`
`char a,b;`
`if (a>b)`
`{`
`}/`

Methods

- **Class** – blueprint of objects
- **Object**-Entity that has **state** and **behaviour**.
Class Pen
`{String shape= “cylinder”}`

State – It includes the different data types ;eg colour,shape of pen.

Functionality- It includes the different functions ;eg writing function of pen.

- Methods- **2** types: **Static & Instance**
- **Static**- invoked by using class
- **Instance/Non static** -invoked by using objects
- **Syntax of method-Static and instance**
Access modifier nonaccess modifier return type method name (parameter list)

```
{//method body  
}
```

- **Access modifier**

- *To set visibility
- *eg:public,protected,default,private
- **Non access modifier**
 - *To set restriction
 - *Static,abstract,final
 - *If **non access modifier is not specified** method is **instance method**
- **Return type**
 - *returns value
 - *void,int ,char

Rules of naming “method”

- Name must start with lower case letter
- If it contains multiple words then first word will have lower case letter and remaining will start with upper case letters.
- Eg:getDataVariable
- The name must not contain any space & should not start with special characters
- It should be a verb eg:main(),calculate()

Rules of naming “class”

- Name must start with upper case letter.
- The name must not contain any space & should not start with special characters
- Eg:AdditionOfTwo
- **Save** the file by giving the class name as file name.
- It should be a noun

Operators

- **Arithmetic**- +,-,/,*
- **Relational**- ==, !=(not equal to), <,>,<=,>=
- **Logical**- &&(AND),||(OR)
- **Assignment**- =,+=,-=,/=,*=
Eg a=a+10 can also be written as a+=10
- **Unary**- ++a(change and use),a++(use and change),--a,a—

28/02/2019

Return Type

- It returns the data from a method to main method.

- The same variable name can also be used in the main method to hold the return value .
- The return type needs to be changed accordingly.

```
class ReturnEx{
public static int add()
{int a=10;
int b=10;
int c;
c=a+b;
return(c);
}
public static void main(String arg[])
{
int r= ReturnEx.add();
System.out.println(r);
System.out.println("End");
}
}
```

- **int c** can also be used instead of **int r** since c is a local variable.

4/03/2019

Parameterised static method/Method with argument

- Method takes an argument from the main and passes a value to the main
- The value passed to the main can also be used as an argument to another method
- The static method is invoked as **classname.methodname()**;

Instance Methods

- **Creation of object** needs to follow the syntax given below:

ClassName Objectname = new ClassName();

```

graph TD
    A[ClassName Objectname = new ClassName();] --> B[Object declaration]
    A --> C[Instantiation]
    A --> D[Initialisation/Constructor]
  
```

Object declaration Instantiation Initialisation/Constructor

- **Instantiation:** It refers to the memory allocation for object, new reserves a memory .
- **Initialisation-** Constructor stores a copy of argument in new.
- **int b** - implies **variable b** declaration
- **Integer b** –implies **Object b** declaration
- Instance method is **invoked** as **Objectname.method Name();**

Constructor

- It is used to **initialise** a object
- **2 types** of constructors
Default constructor-constructor with **no arguments**
Parameterised constructor-constructor **with arguments**
- **Syntax for default constructor**
Access modifier className()
{

}
}
- **Syntax for Parameterised constructor**
Access modifier class name(arguments/parameter list)
{

}

Important

Difference between methods and constructors

Methods	Constructor
<ul style="list-style-type: none"> • Defines behaviour of object • Return type present • It can take any arbitrary name as method name • Method must be created explicitly(we need to create it) 	<ul style="list-style-type: none"> • To initialise an object • Return type absent(will not return value to main method) • It must use class name as method name always • default constructor will be provided by java compiler implicitly(compiler creates the constructor method)

05/03/2019

Important

Default values

- It is **taken** by instance variable only
- Local variable **will not take** instance value
- int-0
short-0
byte-0
long-0L
float-0.0f
double-0.0d
String-null
char-\u000
boolean-false

Static block

- It is used to initialise static variable
- static{
 }
• The value of static variable within the block gets allocated even if the same static variable is assigned a different value outside the static block

Important

Difference between instance and static variable

Instance	Static
<ul style="list-style-type: none">• Each object will have the copy of instance variable, it won't reflect on other objects• It is declared outside method and inside class• The value gets stored in different memory locations in heap or stack	<ul style="list-style-type: none">• It saves memory• If any object changes the value of static variable it will retain the changed value• It is declared with the keyword "static"• The value gets stored in the same memory location of class area after every change in value

Entering value from keyboard

- It is achieved by making use of an inbuilt class called scanner class.
- `import java.util.Scanner;` must be included outside the class
- Syntax is given below
`Scanner s=new Scanner(System.in);`
`int a=s.nextInt();`
- Here instead of int different data types can also be used
- `float e=s.next Float();`
- `double c=s.next Double();`
- `string e=s.next`
OR
`string e=s.nextLine();`

6/03/2019

“this” keyword

- **this** is a reference variable that refers to the **current class object**.
- **To refer instance variable of current class:**by using the statement **this.instance variable** the instance variable can be used as a local variable.
- **To invoke current class method:**A particular method A can be given a higher priority /invoked first before invoking another method B by using the statement **this.method name();** in method B
- **To invoke current class constructors:** A particular constructor A can be given a higher priority /invoked first before invoking another constructor B by using the statement **this(argument of constructor);** in constructor B

Note

- 2 constructors with the **same argument** is **not** possible ie constructor A and constructor B cannot have string argument

- **this** must be included as the first statement whenever it is used


07/03/2019

Method Overloading

- If a class have multiple methods with same name but different parameters it is called **method overloading**.
- If a class have different methods with **same name and arguments of same number and data type** then it is impossible to apply method overloading.
- Method overloading is also **not possible** by **changing** the **return type** of the method alone.
- Method overloading takes place in **static and instance methods**.
- **Use of method overloading**-To increase readability of program.
- **Example** of method overloading

```
Class MethodOverloading
{
    public static void add(int a, int b)
    {
    }
    Public static void add (int c, int d)
    {
    }
}
```

Method overloading **cannot** be achieved



Techniques to perform method overloading

- Change the **number of arguments**.
- Change the **data type**.
- **Example**

```
Class Method Overloading
{public static void add(int a,int b)
{int c=a+b;
System.out.println(c);
}
```

```
Public static void add(int a,int b,int c)//different number of arguments
{int c=a+b+c;
System.out.println(c);
}
```

```
Public static void add (int a,float b)//different data type
```

Method overloading is **possible**



```
{ float c=a+b;  
System.out.println(c);  
}  
}
```

Note

- String grade= "Grade is A" → **Sentence** cannot be returned
return(grade)
- String grade= "Grade " → **word** can be returned
return(grade)
- String grade= "A" → **character** can be returned
return(grade)

08/03/2019

Eclipse

Insatallation

www.eclipse.org → Eclipse IDE for Java Developers

↓
Eclipse photon(4.8)

↓
Eclipse IDE for Java Developers

↓
Windows 64 bit

↓
Download

Create a new java project

File → New → Java project → Assign name → Finish

Create package

src → new → package → Assign name(lower case &no space) → Finish

Create class

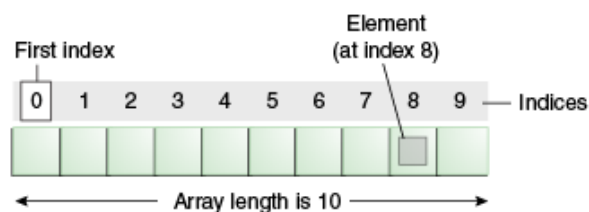
Click on package → class → Assign class name → Tick public static void main(psvm) → Finish

Debugging

- To get inside method- fn&F5
- To go to next line- fn&F6

Array

- It stores elements of **similar** data type
- It is an object
- It is **index based**: First element of array is stored in 0 index
- **Advantage**
Code optimisation: We can retrieve or sort data efficiently.
Random access: We can get any data located at an index position.
- **Disadvantage**
Size: We can store only a fixed set of elements in a java array.
- Arrays are of **2 types**: **Single dimensional** array and **multi dimensional** array
-



Single dimensional array

- **Declaration**

Datatype[] a; → square bracket close to datatype, eg: int[] a;

OR

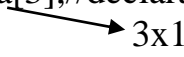
Datatype []a; → square bracket close to variable, eg: int []a;

OR

Datatype a[]; → square bracket close to variable, eg: int a[];

- **Instantiation**

Arrayref variable[] = new datatype[size] → Row size by default (r x 1)

Eg: `int a[]=new int a[3];`//declaration and instantiation


- **Initialisation**

```
a[0]=2;  
a[1]=4;  
a[2]=10;
```

- **Declaration, instantiation &initialisation together**

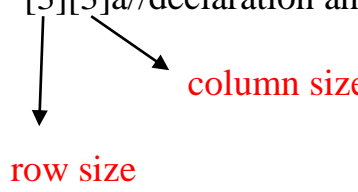
```
int a[]={2,4,10}
```

Multi dimensional array

- **Declaration**

Datatype[] [] array ref variable, eg: `int[] [] a;`
OR
Datatype [][] arrayrefvariable, eg: `int [][]a;`
OR
Datatype arrayrefvariable[], eg: `int a[][];`
OR
Datatype []arrayrefvariable[], eg: `int []a[];`

- **Instantiation**

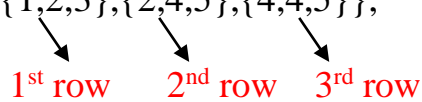
`int [] []a = new int [3][3]a`//declaration and instantiation


- **Initialisation**

```
a[0][0]=2;  
a[0][1]=4;  
a[0][2]=5;
```

- **Declaration, instantiation &initialisation together**

```
int a[][]={{1,2,3},{2,4,5},{4,4,5}};
```



Java String: Non primitive data type

- A java string is same as array of characters
- `char c[]={‘j’, ‘a’, ‘v’, ‘a’};`

`String s=new String(c);`

OR

`String s=new String(“java”);`

OR

`String s= “java”;`

Two ways to create string object

- By new keyword
- By string literal

By string literal

`String s= “java”;`

`String s2= “java”;`

- Here an **object s2** is **not created** as it finds string **object s1** with value “java” hence s2 **acts as reference variable** and points to java. Thus it doesn’t create new instance but it creates a reference variable.
- Each time a string literal is created , JVM checks the string constant pool. If the string already exists in the pool ,a reference to the pooled instance is returned else a new string instance is created and placed in the pool.
- The **usage of string literal** thus makes java more **memory efficient**.

By new keyword

`String a =new String(“java”);`

`String a1=new String(“java”);`

- It creates 2 objects and one reference variable.
- JVM creates a string object in the normal(non pool heap memory) and a literal “java” will be placed in the string constant pool.

13/03/2019

Inheritance

- It is the mechanism in which one class acquires the property of another class i.e a new classes are built based on the existing classes.
- In inheritance one object acquires all the properties and behaviour of the parent object.
- We can reuse **fields(variable)** and **methods** of existing class(parent class).
- Hence inheritance facilitates **code reusability**.
- Inheritance represents **IS-A** relationship(parent-child relationship)
- **Use of inheritance**-code reusability & method overriding
- 3 types- **Single** inheritance, **multilevel** inheritance, **hierarchal** inheritance

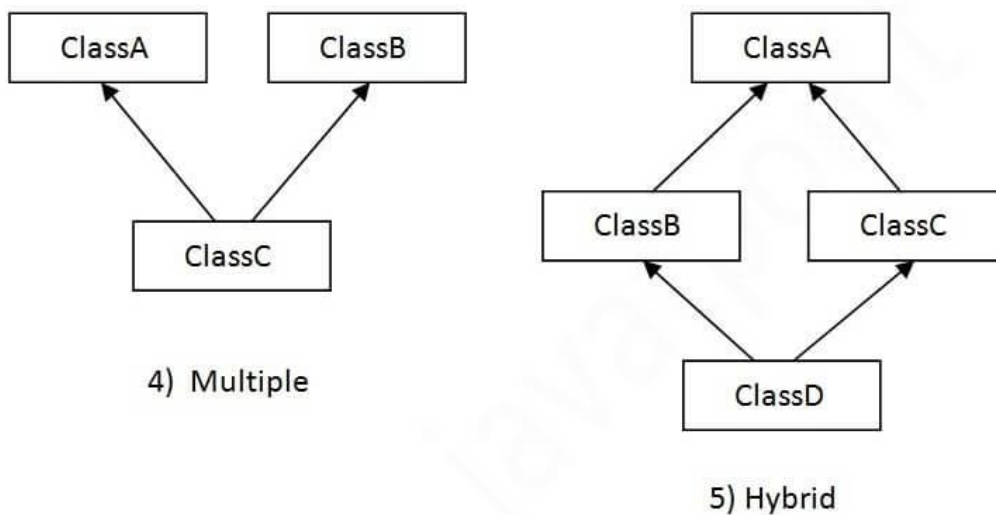
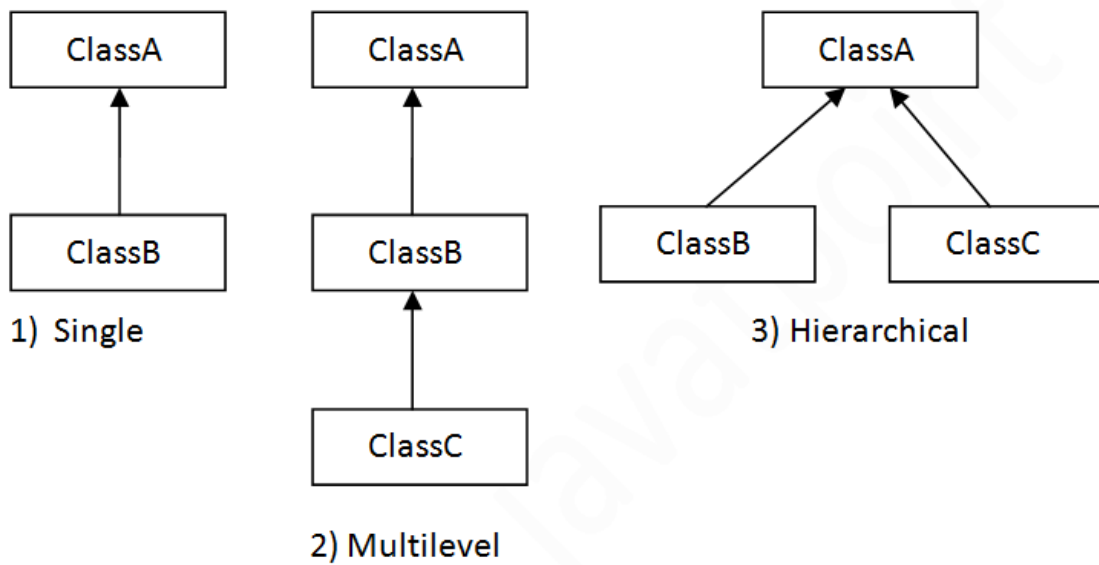
Syntax

```
class child class extends parent class
{
}
}
```

- Keyword “**extends**” is used by child class to inherit features of parent class.
- Child class is also known as **derived class, extended class or subclass**.
- Parent class is also known as **base class or super class**.

Types of inheritance

- Basically 3 types supported by java -single ,multilevel, hierarchal.
- Hybrid and multiple inheritance is **not supported** in java through **class** and is supported only through **interface**.



- **Single inheritance**

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

- **Multilevel inheritance**

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

- **Hierarchical inheritance**

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

Aggregation

- If a class have an **entity reference** of **another class** it is called aggregation.
- **Use of aggregation**-code reusability.
- It represents **HAS-A** relationship.
- Wherever a method needs to be used an object will be created for the specific method with reference to its class

Polymorphism

- Polymorphism is the concept in which a **single action** can be performed in **different ways**.
- **2 types of polymorphism**-compile time polymorphism and run time polymorphism.
- **Example of compile time polymorphism** -method overloading
- **Example of run time polymorphism** -method overriding

Method overriding

- It is the process in which a **child class** will have the **same method** as declared in the **parent class** i.e the **method name, number of arguments and data type** of the argument remains similar to the parent class.
- **Uses of method overriding**
 1. For the specific implementation of a method that has been already declared by its parent class.
 2. For runtime polymorphism
- **Rules for method overriding**
 - 1.The method must have the same name as in the parent class
 - 2.The method must have the same parameter as in the parent class.
 - 3.There must be an IS-A relationship (inheritance).
- **Static method** cannot be overridden since it is invoked by class and not by object. Hence static method cannot be inherited.
- **Private and final method** cannot be overridden as they are local to the class.
- **Java main method cannot be overridden** but **can be overloaded** provided a string argument is passed.

14/03/2019

@override annotation

- @Override annotation is **used over methods** re-defined in a **child class** which might be defined in the parent class. **child class** will have the **same**

method as declared in the **parent class** i.e the **method name, number of arguments and data type** of the argument remains similar to the parent class.

Uses of override annotation

1. **Improves code readability.** One can easily figure out that this method also exists in a super class.
2. **Ensures perfect overriding:** Applying @Override annotation forces the compiler to check if the **arguments** of the overridden method(number and their type) **match with that of super class** . Also, whether its **return type and access modifier matches** with that of **super class**.

If this annotation is missing and the method signature of the overridden method(in child class) does not match with super class method, then it will be considered as **a separate(or new) method which is defined in child class but is not present in parent class.**

3. **Prevents code break due to maintenance activities.** Suppose a method defined in a super class is overridden in a sub-class. Tomorrow, the method signature of super class changes to include one more argument. If the sub-class method has @Override annotation, you will get compiler error and hence code can be corrected. In absence of this annotation, again there will be no compile time error and the method in sub-class will be a separate method which may result in runtime errors.

Important

Difference between method overloading and method overriding

Method overloading	Method overriding
<ul style="list-style-type: none">• It is used to increase readability of the program	<ul style="list-style-type: none">• It is used for the specific implementation of a method already provided by the super class

<ul style="list-style-type: none"> • It is performed within the class • Parameter must be different • method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading but parameter needs to be changed • Example of compile time polymorphism 	<ul style="list-style-type: none"> • It is performed in two classes that have (IS-A) inheritance relation. • Parameter must be same • Return type must be same • Example of run time polymorphism
---	---

“Super” keyword

- **Super** is a reference variable used to refer immediate parent class object.
- **Super is used to refer immediate parent class instance variable:** when the same variable is used in the parent and child class as instance variable. The syntax **super. instance variable** print value of instance variable in parent class.
- **Super can be used to invoke parent class method:** The syntax **super. method name** will invoke the method of the parent class. When the child class has the same method as the parent class and the method is overridden, super keyword can be used to invoke parent class method.
- **Super is used to invoke the parent class constructor:** The syntax is **super (constructor argument).**

Note

- Super is added in each constructor automatically by the compiler if there is no super() or this()

- Super() must always be included as the first statement in method or constructor.

Difference between super and this

super	this
<ul style="list-style-type: none"> • Allows to access public, protected method ,data members, constructor of current class • Cannot access private field and methods 	<ul style="list-style-type: none"> • Allows to access methods, datamembers ,constructors of current class including its own private method and data

15/03/2019

“final” keyword

- It is used to restrict the user from editing the data.
- It can be used along with **variable, method and class**
- **Final variable-value** will be constant and **cannot be changed**. A final variable which is not initialise is called a **blank final variable** and needs to be initialised in a constructor or static block in the case of static method
- **Final method**-If a method uses the keyword final before it then it **cannot be overridden**. If there are 2 methods void run() and final void run() it will result in compile time error.
- **Final class**-If a class uses the keyword final before it then it **cannot be extended** by inheritance.
- **A constructor cannot be final** as it is never inherited.
- **Example**
 public static final int p=2; —————> static final
 public final int p=2; —————> instance final

“static”keyword

- It is used for memory management.
- It is used along with variable, method and class.
- **Static variable**-It gets the **memory only once** in the class area. It can be used to **refer common property** of object such as college name etc
- **Static method**-It gets invoked by the class and it can access static variable and change its value.
- **Restriction on static method**
 - 1.It cannot use non static /instance variable or call non static method directly.
 - 2.this and super cannot be used in static context.
- **Static block** -it is used to initialise a static variable and is executed before main method at the time of loading.

Access modifiers

- **4 types**- private, default, protected, public
- **Private**-accessible only within the class
- **Default**-accessible within the package
- **Protected**-accessible within the package and outside the package through inheritance only
- **Public**-accessible everywhere

Private-default-protected-public



Lowest
visibility



Highest visibility

- **Child class method** must always **have higher visibility** than parent class method.
- Psvm() usually comes along with child class.

Packages

- A java package is a group of similar types of classes, interfaces and sub-packages.
- **2 types**-built in package and user defined package.
- **Advantage of package**
 1. Categorise the classes so that they are easily maintained.
 2. Provides access protection

- 3. Removes name collision
- **Package keyword** is used to create a package
- Example **package** student;
- To **access package** from another package we have the following methods
 1. import package name.*;
 2. import package name .classname;
- **Example of built-in packages** are java, lang, awt, javax, swing, net, io, util, sql etc.

16/03/2019

Encapsulation

- It is the mechanism of **wrapping code**(methods) and **data**(variables) together as a single unit.
- A fully encapsulated class is created by the following steps
 - declare **all its variable as private**
 - use **setter** and **getter methods** to set(**modify**) and get(**view**)the data in it.

Advantages of encapsulation

- By using a getter or setter method it can be made as **read only or write only class**
- **Data hiding**: Other class will not be able to access the data through private variables and hence it acts as protective shield.
- **Control over data**: Setter method gives control over the data as conditional statements can be used within it to select specific data values.
- **Loose coupling** can be achieved by encapsulation.

18/03/2019

Abstraction

- It is the process of **hiding the internal details** and showing only the functionality to the user.
- **2 ways to achieve abstraction**-Abstract class and interface

Abstract class

- A class declared with **abstract keyword** is known as abstract class.

- It provides **0-100% data hiding** based on the methods used within the class
- The methods of an abstract class can be **abstract or non abstract methods**. It provides **100% data hiding** if **all the methods are abstract**.
- Abstract class **cannot** be **instantiated**.
- Abstract methods can be defined **only in abstract class**.
- Abstract class will have **constructor** and **static method**.
- The **definition** of all the abstract method must be **provided** in the extended **class which is not abstract**.
- The abstract method will just include the **signature of the method** without its definition
- Abstract class can have **final methods**.

- **Syntax**

```
abstract class classname
{
    abstract returntype methodname()
    {
    }
}
```

- **Example**

```
public abstract class Test
{
    public void hello();//non abstract method
    public void abstract hey();//abstract method
    Test();//constructor
    {
    }
}

Class ABC extends Test
{public void hey()
{//definition of abstract method
}
public void hello()
{//definition
}
public static void main()
{
}
```

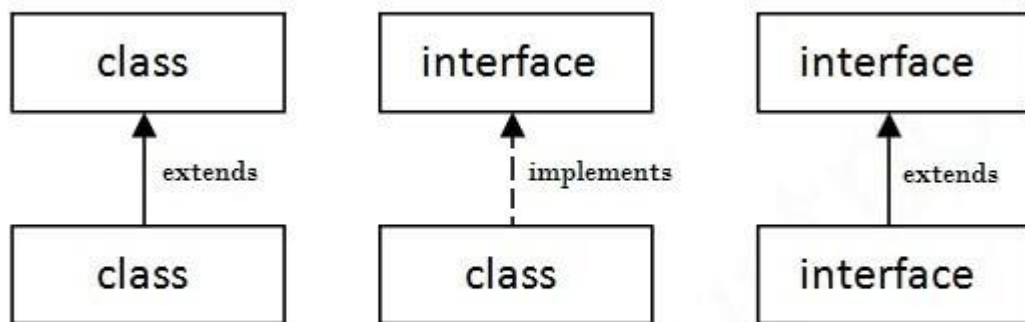
Interface

- It is a blueprint of class.
- In interface all **methods** will be **public & abstract** without method body and **variables** will be **public static and final** by default.
- It does not include **instantiation** and **constructor**.
- Java 8 supports **default and static methods** & Java 9 supports **private methods** also in interface.
- **Uses of interface**
 1. To achieve **100% data hiding** since all methods are abstract here.
 2. It helps to achieve **multiple inheritance**.
 3. It helps to achieve **loose coupling**.

Note

Loose coupling: It means reducing dependencies of a class that use different class directly .

- **Syntax**
interface interface name{
//abstract methods
//public static final variable
}
}
- **Relation between class and interface**



- Class can be either **abstract or nonabstract** in the above figure. In both case the keyword “extends” and “implements” used in figure holds.

- **Example**

Interface Test ↗ constants
 { **public static final** a=10;
 public abstract hello();
 }

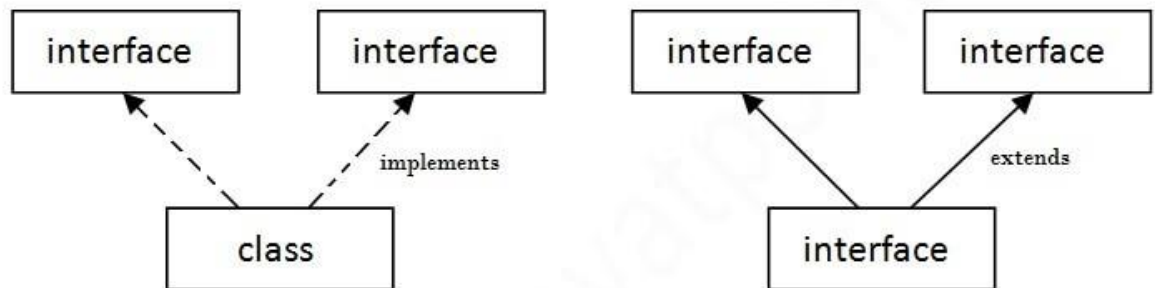
Class ABC implements Test
 {public void hello()
 { System.out.println("hello");
 }
 }

Public static void main()
 { Test ob=**new ABC()**;
 ob.hello();
 }

Instantiation/constructor is possible only for ABC and not Test as interface does not support it.

- **Multiple inheritance**

If a **class** implements **multiple interfaces**, or an **interface** extends **multiple interfaces**, it is known as multiple inheritance.



Multiple Inheritance in Java

- **Example**

```

interface Printable
{
void print();
}
interface Showable
  
```

```

{
void show();
}
class A7 implements Printable,Showable{
public void print()
{System.out.println("Hello");
}
public void show()
{System.out.println("Welcome");
}
}

```

```

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
}

```

Difference between abstract class and interface

abstract	interface
<ul style="list-style-type: none"> • It can have abstract and non abstract methods. • It doesn't support multiple inheritance. • It can have static,non static,final and non final variable. • Abstract class can provide implementation of interface. Eg :abstract class Classname implements interface • Abstract keyword is used to declare abstract class • An abstract class can extend another Java class and implement multiple Java interfaces. 	<ul style="list-style-type: none"> • It can have only abstract methods. • It supports multiple inheritance. • It can have only static and final variable. • Interface cannot provide implementation of abstract class Eg: Interface interfacename implements classname (not possible) • Interface keyword is used to declare interface. • An interface can extend another Java interface only

<ul style="list-style-type: none"> • A abstract class can have class members like private, protected, etc 	<ul style="list-style-type: none"> • Members of a interface are public by default.
--	---

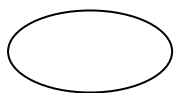
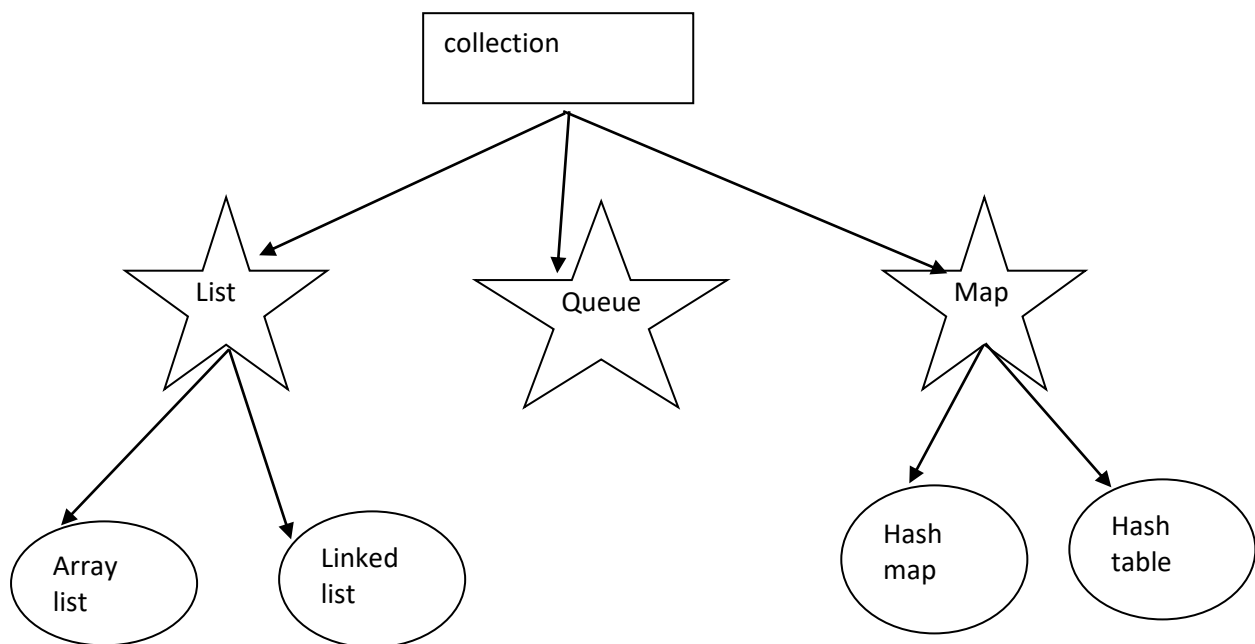
Difference between abstract and final

Abstract	final
<ul style="list-style-type: none"> • It can be inherited to subclasses • It can be altered in subclass • It cannot be instantiated • It can contain abstract methods • Extra functionality to the methods can be added • All methods need not be implemented 	<ul style="list-style-type: none"> • cannot be inherited • final method cant be overridden • It can be instantiated • It must not include abstract methods • No extra functionality can be added • All methods should be implemented

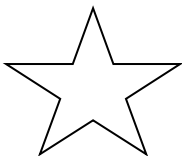
19/03/2019

Collections

- Collection in Java is a **framework** that provides an architecture to store and manipulate the **group of objects**.
- It includes **interfaces**(list, queue, map) and **classes**(array list, linked list etc)
-



Represents the classes of framework



Represents the interface of framework

Note

jar-java archive file

List interface

- It includes list type data structure in which we can **store** the ordered collection of objects. It **can have duplicate values**.
- **Syntax**
 1. List <data-type> a1= **new** ArrayList();
 2. List <data-type> a2 = **new** LinkedList()
- It is mainly used to insert, delete, and access the elements from the list

Array list

- It uses a dynamic array to store the element of different data types
- It maintains the insertion order based on array index.
- **Syntax**
ArrayList<String> a1=**new** ArrayList<String>();
↓
Generics/data type → JDK 1.5 feature
OR
- **List**<String> a1=**new** ArrayList<String>();
OR
- List<String> a1=**new** ArrayList(); → generic on RHS absent-**JDK 1.7 feature**
- a1.add(20) → places the value 20 in the next index position of array
- a1.remove(2) → removes the value stored in 2nd index position of array
- a1.size() → returns size of array
- a2.addall(a1) → copies all values of a1 to a2

Advanced for loop

datatype
↑
for(Integer i:a1)
{ system.out.println(a1.add(i))
} //prints the values added to array

- If a1.add(20) representing integer datatype and a1.add("abc") representing string data type needs to be printed with the same for loop then the following syntax is used :


```
for(object i:a1)
{system.out.println(a1.add(i))
} //prints the values added to array
```

Linked list

- It performs the same operations as an array list
- However **linked list is less efficient than array list** due to thread issues
- Syntax

```
LinkedList<String> al=new LinkedList<String>();
OR
List<String> al=new LinkedList<String>();
OR
LinkedList<String> al=new LinkedList();
```

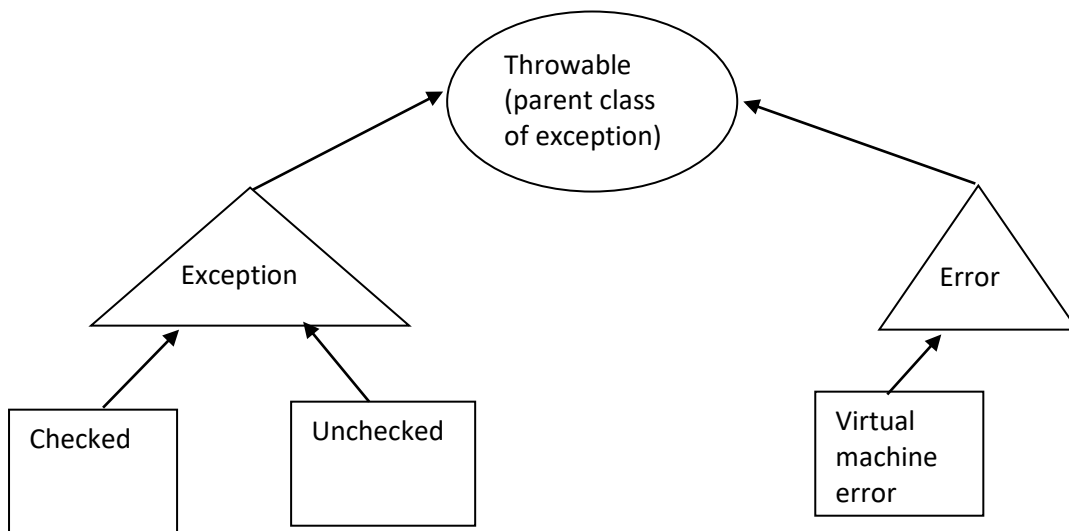
21/03/2019

Exception Handling

- It is an **unwanted condition** that disrupts the normal flow of program.
- It **can be handled**
- It is a **class**
- 2 types of exception-**Checked and Unchecked**
- **Checked/compile time exception** -program doesn't run due to **external error** . It can be found and corrected at **compile time**
Eg: Input-output exception-File cannot be read as path is different
SQL exception-error in data base
- **Unchecked/runtime exception**-program doesn't run due to **error created by the programmer**. It cannot be corrected in compile time as issues occur in run time.It is corrected at **run time**
Eg: Arithmetic exception
Array out of bound
- Compile time exception is **mandatory** and runtime exception is **optional**
- Class Exception is the **parent class** and all other classes under it are taken as child class
Eg:Class SQL exception extends Exception
Class File not found exception extends Exception

↓
Child class

↓
Parent class



Java try-catch block

- Java **try** block is used to enclose the code that might throw an exception
- It must be **used within the method**
- If an exception occurs at the particular statement of try block, **the rest of the block code will not execute**. So, it is better not to keep the code in try block that will not throw an exception.
- **Java try block** must be followed by either **catch or finally block**.
- **Java catch block** is used to handle the Exception by declaring the type of exception within the parameter
- The catch block must be used **after the try block** only
- A try statement can have **multiple catch blocks**.
- A **try catch** block comes only within **psvm**.
- **JVM has a default try catch block** which handles exception in case the exception is not handled

- **Syntax**

try

{

//code that may throw an exception

}**catch**(Exception_class_Name ref){}

OR

```
try{  
    //code that may throw an exception  
}finally{ }
```

- **Example**

```
public class TryCatchExample2 {
```

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

```
        {  
            int data=50/0; //may throw exception  
            System.out.println("exception")  
        }
```

Won't get printed since
it is inside try block

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

Can use **Exception** (parent
Class) instead
Of **arithmetic
exception**(child class) in
that case it handles all
exception

```
        System.out.println("rest of the code");  
    }
```

gets printed as it is
Outside the try
Block & gets
Printed even in the
Presence of
exception

```
}
```

Multiple catch block

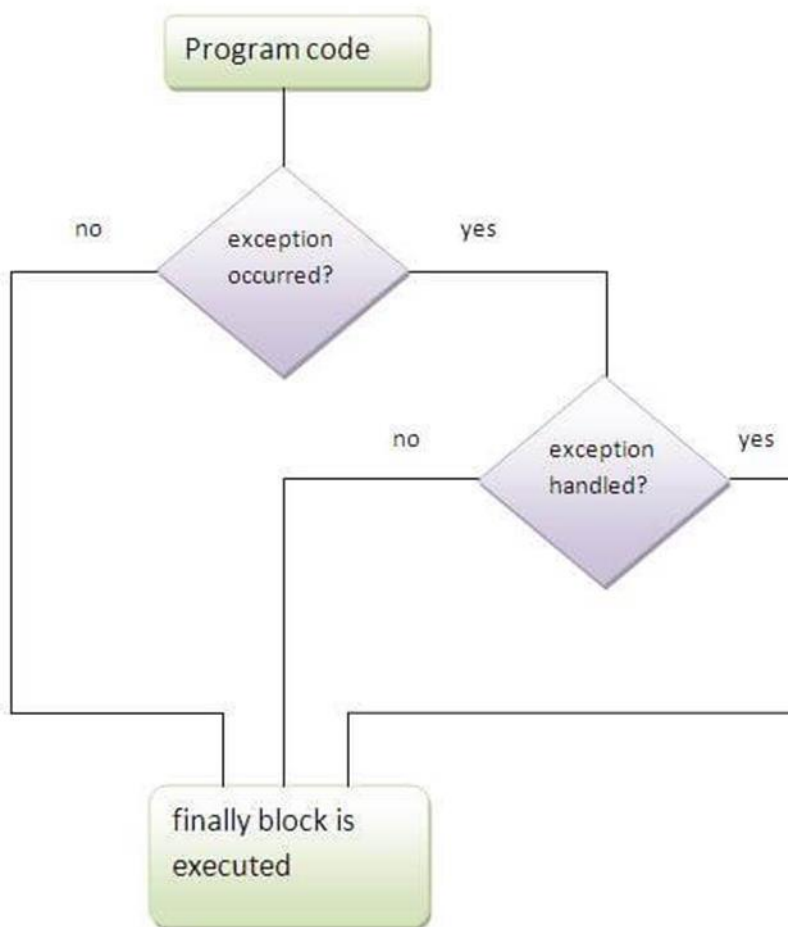
- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from **most specific to most general**, i.e. catch for `ArithmeticException` must come before catch for `Exception`, else it results in compile time error
- In case there are **two exceptions** within the **try** block, the exception which is listed first in the catch block alone will be handled.

- In case there are two exceptions and **both are not listed in catch block** then the catch block with the general class exception gets invoked.

Java nested try block

- Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. Here we make use of nested try block.

Java finally block



- Java finally block is a block that is used to execute important code such as closing connection, stream etc
- Finally ensures **mandatory execution** of code within it.
- Java finally block is **always executed whether exception is handled or not.**
- Java finally block **follows try or catch block**

- If you don't handle exception, before terminating the program, **JVM executes finally block(if any).**
- **Use of finally**-to close a file or connection

Note

- For each try block there can be zero or more catch blocks, but only one finally block.
- The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

Java throw keyword

- throw keyword is used to throw an exception
- It can be used to throw checked or unchecked exception
- **Syntax**
throw new exception;
- It is used mainly for **custom exceptions**(exceptions created by us)
- **Example**

Exception defined by us(custom exception)

```

class Mickeymouse exception extends Exception
{
void validate(int age){
    if(age<18)
        throw new Mickeymouse Exception("not valid");
    else
        System.out.println("welcome to vote");
}
public static void main(String args[]){
    Mickeymouse ob=new Mickeymouse();
    ob.validate(13);
    System.out.println("rest of the code...");
}
}
  
```

Output

Exception in thread main java.lang.**MickeymouseException**:not valid

Java throws keyword

- It is used to **declare an exception**

- It gives an information to the programmer that there may occur an exception
- It is mainly used to handle checked exception as unchecked exception handling is programmers responsibility
- **Syntax**

```
returntype  methodname() throws exceptionclassname
{
//method code
}
```

- Exception is thrown from method to main and main to JVM
- When it is thrown **from method**, try catch block within **psvm** should handle it
- When thrown **from main** ,**JVM handles** it by using the default try catch block

- **Example 1**

```
Class Test{
Public void hello() throws IO Exception file
{
}
Public void print()
{
}
Psvm( )
{ Test ob =new Test();
  Ob.print();
  try
  {
    Ob.hello();
  }catch {Exception e}
}
```

Method throws an exception

throws keyword absent ,won't check for exception in this method

Psvm throws exception handled by JVM

- **Example 2**

```
Public static void main ()throws IO Exception file
{
}
```

Important

Difference between throw and throws

throw	throws
<ul style="list-style-type: none"> • Java throw keyword is used to explicitly throw an exception. 	<ul style="list-style-type: none"> • Java throws keyword is used to declare an exception.

<ul style="list-style-type: none"> • Checked exception cannot be propagated using throw only. • Throw is followed by an instance. • Throw is used within the method. 	<ul style="list-style-type: none"> • Checked exception can be propagated with throws. • Throws is followed by class. • Throws is used with the method signature.
---	---



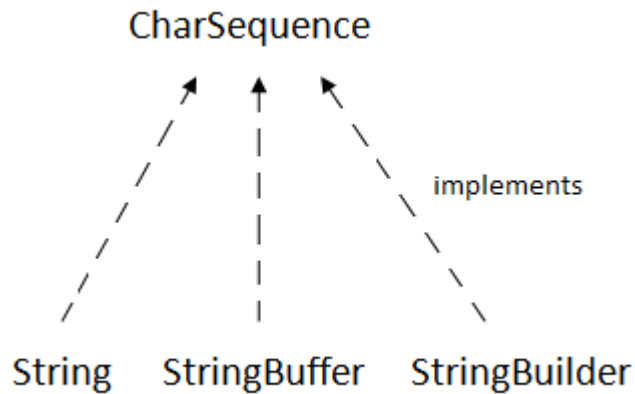
Difference between final, finally and finalized

final	finally	finalize
<ul style="list-style-type: none"> • Final is used to apply restrictions on class, method and variable. • Final class can't be inherited • final method can't be overridden • final variable value can't be changed • Final is a keyword. 	<ul style="list-style-type: none"> • Finally is used to place important code, it will be executed whether exception is handled or not. • Finally is a block 	<ul style="list-style-type: none"> • Finalize is used to perform clean up processing just before object is garbage collected. • Finalize is a method.

22/03/2019

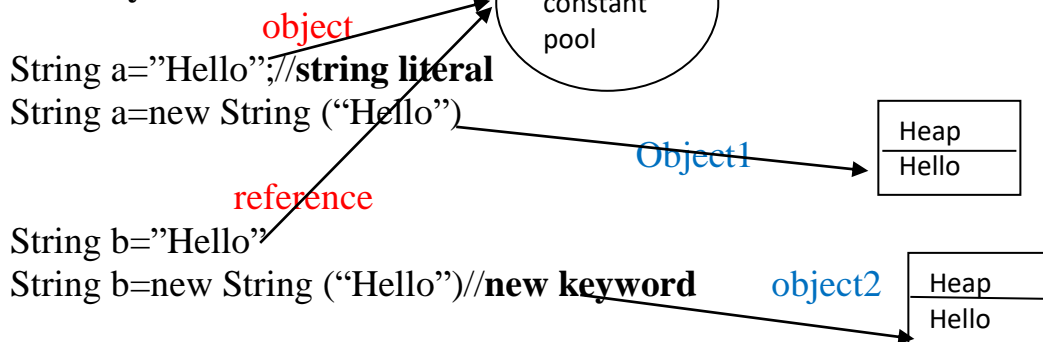
String class, String buffer class and string builder class

- String is basically an **object** that represents **sequence of char values**.
- An array of characters works same as Java string



- We can create strings in java by using these three classes.
- The **CharSequence interface** is used to represent the sequence of characters
- `String`, `StringBuffer` and `StringBuilder` classes implement it.
- The **Java String is immutable** which means it cannot be changed.
- Whenever we change any string, **a new instance** is created.

Memory allocation



Concatenation of string

- 2 methods- By + (string concatenation) operator
- By `concat()` method
- String concatenation is implemented through the **StringBuilder (or StringBuffer) class**

- String concatenation operator produces a new string by appending the **second** operand onto **the end of the first** operand
- The string concatenation operator can concat **not only string but primitive values also**

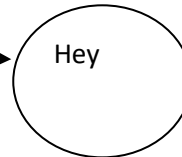
Example of + (string concatenation) operator

- String a=50+40+"hello"+20+20;
System.out.println(a);
Output:90hello2020

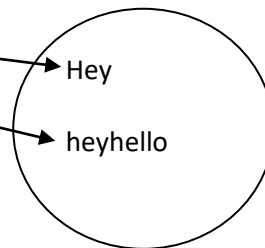
↓
(50+40)

Example of concat method

- String s="hey";
s.concat("hello")
system.out.println(a)
output:hey//both s points to same location



- String s="hey";
s=s.concat("hello")
system.out.println(a)
output:heyhello
//s points to different location



String buffer class

- Java StringBuffer class is used to create mutable (modifiable) string.
- Java StringBuffer class is **thread-safe** i.e. multiple threads cannot access it simultaneously and **synchronised**
- 4 methods
 - ***append**-The append() method concatenates the given argument with this string.
 - ***insert**-The insert() method inserts the given string with this string at the given position.
 - ***delete**-The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

***replace**-The replace() method replaces the given string from the specified beginIndex and endIndex.

Example

```
String buffer s="hello";  
s.append("hey");output=hellohey  
s.replace(1,3,"java");output=hjavallo  
s.insert(1,"java");output=hjavaello  
s.delete(1,3)output=hlo
```

String builder class

- Java StringBuilder class is used to create **mutable (modifiable) string**.
- The Java StringBuilder class is same as StringBuffer class except that it is **non-synchronized and not thread safe**.
- It includes the same **4 methods** given above:append(),delete(),insert(),replace()



Difference between string buffer and string builder

String buffer	String builder
<ul style="list-style-type: none">• StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.• StringBuffer is less efficient than StringBuilder.	<ul style="list-style-type: none">• StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.• StringBuilder is more efficient than StringBuffer.



Difference between string and string buffer

String	String buffer
<ul style="list-style-type: none">• String class is immutable.• String is slow and consumes more memory when you concat too many strings because every time it creates new instance.• String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	<ul style="list-style-type: none">• StringBuffer class is mutable.• StringBuffer is fast and consumes less memory when you concat strings.• StringBuffer class doesn't override the equals() method of Object class.

Iterator interface

- Interface with methods hasNext(),next(),remove()
- Iterator permits the caller to remove the given elements from the specified collection during the iteration of the elements.
- **hasNext()**-Returns a true value if the more number of elements are encountered during iteration.
- **next()**-Returns the next specified element during the iteration.
- **remove()**-Removes the last element from the collection as provided by the iterator.

- **Example**

```
List<String> list = new LinkedList<>();  
list.add("Welcome");  
list.add("to");  
list.add("our");  
list.add("website");
```

```
Iterator<String> itr = list.iterator();
```



Class arraylist has a method `iterate()` which returns an **object iterator itr** referring to the interface iterator

```
//Returns true if there are more number of elements.
```

```
while(itr.hasNext()) {
```

```
    //Returns the next element.
```

```
    System.out.println(itr.next());
```

```
}
```

```
    //Removes the last element.
```

```
itr.remove();
```

