Java

Java

Data Types

Primitive

1) Integer type
   - byte ⇒ e.g. 10
   - Short ⇒ e.g. 98833 - 2 bytes
   - Int ⇒ e.g. 9883333 - 4 bytes
   - long ⇒ e.g. 9883333333 - 8 bytes

2) Decimal Type
   - float ⇒ e.g. 12.345
   - double ⇒ e.g. 1.87129

3) Character Type
   - char ⇒ e.g. 'A'
   - boolean ⇒ eg. true, false

Non primitive

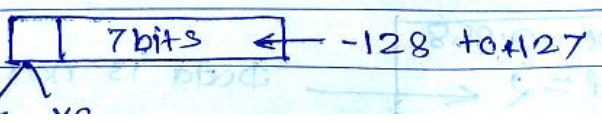
string for sentences and words.

Binary System -

1 Bit = 0, 1 — 2 outcomes
2 Bit = 2^2 = 4 outcomes
n Bit = 2^n = outcomes

1 Byte = 8 Bits = 2^8 = 256 outcomes.

Bit ⇒ 1st 2 outcomes out of 256 reserved for Sign (+ve, -ve)
Remaining 7 Bits gives -128 to 127.
as $2^7 = 128$



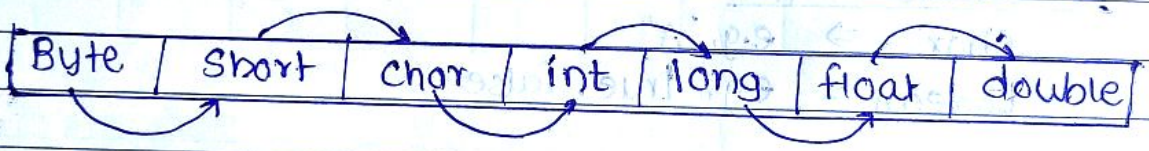
Variables

`DataType variable = value`

e.g. `int a = 10;`
`boolean a = false;`
`char a = 'A'` ⇐ single quote for char double for string.

Type Casting

Converting data type to another form.



boolean not used here

Types of Type casting

1) **Widening / Automatic Type Casting** - In sequence as shown above

```
int x = 4;
double y = x;
```

`y = 4`

Casting also happens when byte type enter as parameter in method who accepts int data

2) **Narrowing / manual Type casting** - In reverse order large data into smaller data type.

```
double x = 2.888;
int y = (int) x;
```

`x = 2.888`
`y = 2` ← Data is narrowed.

Operators

1) Arithmetic - +, -, *, /, %, ++, --

(1)

2) Bitwise -

NOT (!)

X	X'
0	1
1	0

And (&)

X	Y	X&Y
1	0	0
0	1	0
1	1	1
0	0	0

OR (||)

X	Y	X||Y
1	0	1
0	1	1
1	1	1
0	0	0

XOR (^)

X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

Right Shift 1101 ⇒ 0110 ⇒ 0011 ⇒ 0001

left Shift 1101 ⇒ 1010 ⇒ 0100 ⇒ 1000

Right Shift is dividing by 2 & left is multiplying by 2 the int type value hence recieving any floor value.

e.g. int a = 13
int b = a>>2; // two Right shifts

b = 3

3) Assignment operator
'='

a = 2; 2 assigned to a

a += 2 ⇒ a = a+2

a &= 2 ⇒ a = a&2

a ||= 2 ⇒ a = a||2

a %= 2 ⇒ a = a%2

a *= 2 ⇒ a = a*2

4) Comparison Operator

They Returns boolean Type value.

<, >, <=, >=, (==), (!=)

Binary Numbers

Decimal

124

1×10^2 2×10^1 4×10^0

Binary

1 0 0 1 1 0 1

$2^6\times1$ $2^5\times0$ $2^4\times0$ $2^3\times1$ $2^2\times1$ $2^1\times0$ $2^0\times1$

$= 64+0+0+8+4+0+1$

1) Binary to decimal By above method

$(1001101)_2 = (77)_{10}$

2) Decimal to Binary by divisdation.

2	74	0
2	37	1
2	18	0
2	9	1
2	4	0
2	2	0
2	1	

$(74)_{10}$

$\Downarrow$

$(1\ 0\ 0\ 1\ 0\ 1\ 0)_2$

⇐ proceed till one

Operators on Binary

4 & 6

100 110

4	6	4 & 6
1	1	1
0	1	0
0	0	0

100 gives 4

$\therefore$ 4 & 6 = 4

1) 45 & 10

2) 45 ∧ 10

3) 45 || 10

45 = 101101

10 = 1010

= 001010

2	45	1
2	22	0
2	11	1
2	5	1
2	2	0
	1	

2	10	0
2	5	1
2	2	0
	1	

45	10	&
1	0	0
0	0	0
1	1	1
1	0	0
0	1	0
1	0	0

45	10	||
1	0	1
0	0	0
1	1	1
1	0	1
0	1	1
1	0	1

45	10	∧
1	0	1
0	0	0
1	1	0
1	0	1
0	1	1
1	0	1

= 1000

= 8

= 101111

= 47

= 100111

= 39

Q 1 0 1 1 1 1

$= 2^5 + 0 + 2^3 + 2^2 + 2^1 + 2^0$

$= 32 + 0 + 8 + 4 + 2 + 1$

$= 47$

Keep sequence of $\boxed{2^n}$ in mind and keep adding with skipping whereas $\boxed{0}$ in Binary.

Conditional Statement

1) if, else if, else

```
if (condition) { //code }
```

↑ Boolean type return with conditional operators

```
else if (condition 2) { //code }

else { //code }
```

2) Ternary operator (shortcut to if else):

```
variable = (condition) ? code : code ;
```

↑ If True ↑ If false

e.g.
```
int a = 15, b = 8 ;
int max ,
    max = a>b ? a : b ;
```

&&, ||, ! logical operators - if 1st condn satisfy then go next

& , | Bitwise Operators - check both at a time

(|| or |) gives same results.

Not operator (!) converts boolean values

Nested if else can also be used as

```
if (condition1) {
        if (condition2){ // code }
    else if (condition3) { //code}
else {code}
```

Nested ternary operator

```
variable = (condition1) ? (condition2)? code : code : conditon3 ? c : c;
```

3) Switch

```
int a = 3;
switch (a) {
    case 1:
    case 2:
    case 3: sysout ("Bad")
            break;
    case4: break;
    case5: break;
}
```

Bad

Loops

1) For

```
for ( int i = 0 ; i<5 ; i++){
      statement1  statement2  statement3
      initialisation  condition  Reinitialisation.
        //code to execute
}
```

nested for loops can also be used.

2) do while loop

```
do {
    //code to execute
} while ( condition);
```

Irrespective of condition code run once then condition is checked

↑ if condition true loop is repeated

3) While loop

```
while (condition) {
    //code
}
```

↑ If condition true then enter to execute code

4) for each loop - used for arrays.

```
for (datatype variable : array name variable){
        //code to execute
}
```

eg

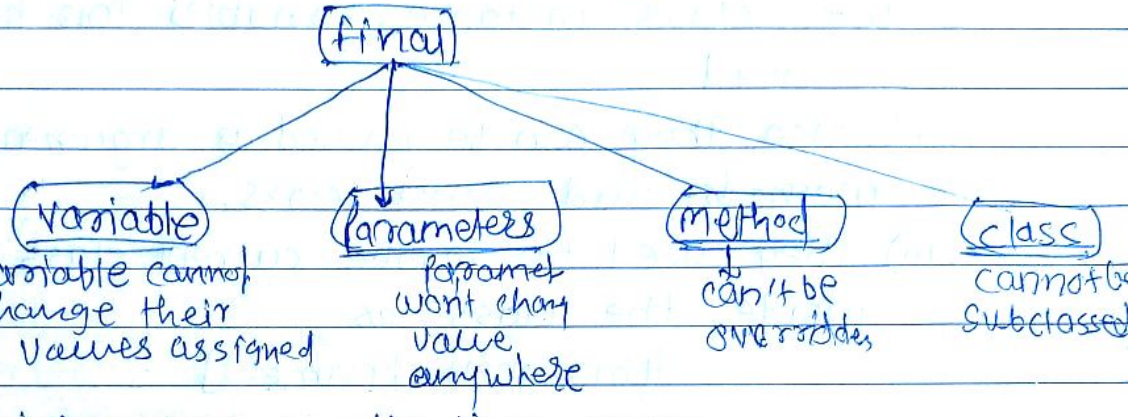
```
for (int i : a){
        code
}
```

Q. what is static variable? Q what is static block?
Q limitations of static methods?
Q why main method is static → Because JVM will need to create object and that will take memory.

Keywords

~~Q what is final variable/class/constructor/method?~~

1) final - anything declared final remains constants its values cannot be changed.



overriding gives compile time error.
Interfaces, constructor,

2) static -

static

- variables - takes memory once and shares variable to every instance without creating actual memory space each time for that variable.

static int n = 10

share share

object1 object2

saves memory

- method -) static method can be invoked without creating instances.

int a = Class.method();

2) this, super cannot use in static method

3) static method cannot use nonstatic data or nonstatic methods directly.

- block - static block inside class will excuted once before main method

Imp.

we can excute program without main method as using static block (But in lower versions of Java).

Q can we excute program without static main method?

Q. what if we remove 'static' from main method?

without static in main method compiles throws [No such method Error]

without static JVM has to create objects to excute which takes memory hence we use 'static' in main

3) this - i) when variables passed into argument are same as variables of class then to refer the class instance variables this keyword is used.

ii) Also they can be passed as arguments in methods and constructors.

iii) They used to invoke current class methods inside the class as

this.method name()

Q. what is this keyword? uses } can both used simu

Q. what is super keyword? uses } can they give constru chaining?

4) super - i) keyword used to refer immediate parent class object when we create object of class, object of parent class created implicitely which is refered by "super".

uses - i) To invoke immediate parent class methods, constructors and refer variables.

Difference

this	super
i) used to refere instance variable of same class	i) used to initialise parent cla variable in child class constru
ii) points to current class context	ii) points to parent class context

Similarities

Both used for constructor chaining.

Both must be first statement inside constructor otherwise error is thrown.

As they should be first statement always both cannot used simultaneously.

Q. what is instance of operator.

5) instanceOf (operator not keyword) - comparison operator returns true or false compares type with instance.

Class dog extends Animal.

In main method

dog d = new dog();

x = d instanceof dog // true.

Q what are access specifiers in java.

Access modifiers	within class Y/N	within package Y/N	outside pac by subclass Y/N	outside package Y/N

Access Modifier keywords / Specifiers

1) private - access only within class can't accessed outside the class.

Y	N	N	N

2) Default - access level is only within the package If we don't specifie then its access level is defaut.

Y	Y	N	N

3) Protected - accessible by other classes outside package if they inherites the given class otherwise not.

Y	Y	Y	N

4) Public - accessible from anywhere anyclass & any package.

Y	Y	Y	Y

Q. What is OOP Paradigm

OOP

OOP = Paradigm - we have classes which have states and behaviors and we produce objects based on it. It supports following properties.

1) Polymorphism
2) Inheritance
3) Abstraction
4) Encapsulation.

Protected:

Protected variable, methods are accessible by child class only.

Polymorphism (more than one form)

Polymorphism means having many forms. e.g. ability of message to be displayed in different forms.

Types of polymorphism in JAVA

1) Compile time 2) RunTime

1) Compile time polymorphism - It is a static polymorphism this type of polymorphism achieved by function Overloading or operator Overloading.

i) function overloading/Method overloading - when there is multiple functions with same name but parameters are different then its method overloading. we can change either number or type of arguments like follow.

In multiply class

```
① static int multiply (int a, int b) {
        return (a * b);
  }

② static int multiply (int a, int b, int c) {
        return (a*b*c);
  }

③ static int multiply (String a, String b) {
        sysout ("a*b");
  }
```

In main class

```
sysout ( multiply.multiply (10, 20));
Sysout ( multiply.multiply (10, 10, 10));
sysout ( multiply.multiply (5, 5));
```

Output

200
1000
5*5

Q. what is overriding methods? can be static methods overridden ⇒ No

Q can we overrid overloaded method ⇒ Yes

Q can private methods overrided?

i) Operator Overloading - we can make use of '+' operator for concating string and also for addition of integer etc.

```
In class operator{
    Void operator(String a, String b){
                String s = String a + String b;
            sysout("concated strings"+s);
            }
    Void operator(int a, int b){
                int c = a+b;
            Sysout("addition is -"+c);
```

In main class

```
Sysout(Operator.operator(Shubham, Sawant);
Sysout(10, 20);
```

Output

concated Strings Shubham_Sawant addition is 30

2) Runtime Polymorphism Dynamic method dispatch Polymorphism in which function is called to overriden method at runtime achieved by method overridding.

i) method overriding - when Base & derived classes has same methods but in derived class it is implement differently then called as overridding the method in Base class by derived class.

In class Base {methods given as.

```
Void Print(){ SysOut("Base method");}
```

In subclass Derived1 method given is

```
Void print() {sysout("Derived1 method");}
```

In subclass Derived2 method given as

```
Void print() {sysout("Derived2 method");}
```

In main class

```
Base a = new Derived1();
Base b = new Derived2();
a.print();
b.print();
```

Output

```
Derived1 method
Derived2 method
```

* Desicion of which method of overidden methods is to be executed is made during running of code hence called runtime polymorphism.
where as in other formats decision made during compilation i.e (during writting code comilation of code take place simultanously)

* Static methods not overridden
* overlooded methods overrided
* private method cannot overridden as they cannot used outside class.

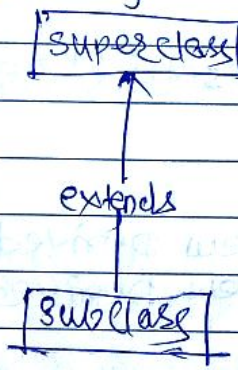
Q. what is runtime polymorphism

Dynamic Method Dispatch / Runtime Polymorphism

1) When overridden method is called through refference of superclass Java determines which version of method is to be executed based on type of object being refferred to at time of call. This is made at Run time

Upcasting

SuperClass obj = new Subclass



Hence different objects of various classes reffered by refferance of superClass. By this different version of ~~methods executed~~.

```
class A{
  Void m1 (){ sysout ("A method");}
  }
Class B {extends A;{
  Void m1 (){ sysout ("B method");}
  }
Class C extends A;{
  Void m1 (){sysout("C method");}
Class main{
      public static void main (String[] args){
      A a = new A();
      B b = new B();
      C c = new C();
```

```
A ref;
ref = a;
ref.m1();
ref = b;
ref.m1();
ref = c;
ref.m1();
```

Output

```
A method
B method
C method
```

2) We can override the methods only But we can't override the variables and data members. hence runtime polymorphism cannot be done by Data members & variables

```
class A{
    int x = 10; }
class B extends A{
    int x = 20; }
public class Test{
        public static void main (String[] args){
    A a = new B();
    sysout (a.x);
}
}
```

Output

```
10
```

hence variable retained from superclass and not overridden.

Q. Difference of compile time & Runtime.

	Compile time	Runtime
i)	known as static binding, early binding or overloading	i) known as dynamic binding overriding.
ii)	overloading of methods	ii) overriding of methods
iii)	fast excution as type of object determined at compile time.	iii) slow excution as type of object determined ~~at~~ runtime.

Reallife e.g.

The man can be having different rolls at the same time as he is son, father, employee etc has different characteristics similarly methods can be having different structure and applicabi

Q. what is inheritance? types? Advantages.

Inheritance

Inheritance - Mechanism through which subclass gets or inherits features (field and methods) of super class

a) Superclass - Base/Parent class whose features inherited.
b) Sub class - Derived/extended/child class who inherites feature
c) Reusability - This concept allow us to use method and fields of existing class for creation of new. hence methods and fields reused.

follow 'is a' relationship e.g. car is a vehicle.

* Inheritance property used with (extends) keywords.

When object of subclass created, a copy of all methods & field of Superclass aquire memory in this object, thus by using object of subclass we can also access the member of a Superclass.

When object of subclass created it is a object that subclas only not of superclass. only objects of subclass created that has super class variables.

e.g.

```
Class Fruit {
    Public Fruit() { Sysout (this.hashCode());
                     sysout (this.getClass().getName()); } }

Class Apple extends Fruit {
    Public Apple() { sysout (this.hashCode());
                     sysout (this.hashCode + "_" + this.
                                                  super.hashCod
                     sysout (this.getclass().getName())
                     sysout (super.getclass().getName()
                   } }

public class Test
    {public static void main (String[] arg)
        { Apple myApple = new Aple();
        } }
```

Output

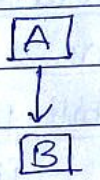
```
366712642
Apple
366712642
366712642  366712642
Apple  Apple
```

hence object points to subclass not superclass.

Types of Inheritance

1) Single Inheritance - Subclass inherites the features of one super

```
Class one{
 public void print(){ sysout("Shubham");
                    }}
Class Two extends One {
 public void printat(){ sysout("sawant");
                    }}
public class Main
          { public static void main(String[] arg) {
    Two g = new Two();
     g.Print();
     g.Printat();
    }}
```



Single Inheritance

Output

```
Shubham
sawant
```

2) Multilevel Inheritance - one class become base class for one class and that derived class acts as base class for other.

But child class can't access grandparent class directly.

```
class GrandParents {
        public void Print(){
              Sysout("GrandParent");
        }}
Class parent extends Grandparents{
        public void print(){
              Sysout("Parent");
        }}
Class child extends parent{ super.super.print()
        Public void print(){
```

Trying to access GrandParents class

```
}}
public class main { public static void main (string[] arg) {
        child c = new child();
          c.print();
}}
```

output

Compile Error - child class cannot access grandparent method and fields as allowed in C++ by (::) operator.

In JAVA we can access Grand Parents members though parent only as follows.

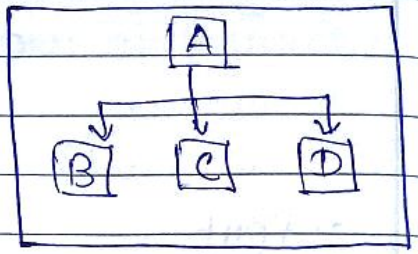
```
class Grandparent {
      Public void print() { sysout ("Grandparent");
                          }}
class Parent extends Grandparent{
      Public void Print() { sysout ("Parent");
                            super.print();
                          }}
class child extends parent{
      public void print() { sysout ("child");
                            super.print
                          }}
Public class main {
      Public static void main (String[] args) {
            child c = new child();
              c.print();
      }}
```

output

Grand Parent
parent
child

hence we can access grandparent through parent by super keyword

3) Hierarchical Inheritance - one class becomes base for many subclasses.




```
class one {
public void print(){ sysout("Shubham");
        }}
class Two extends one {
  Public void print(){ sysout("Sawant");
        }}
class Three extends one{
  Public void print(){ sysout(
        }}
class main
    Public static void main (String[] args){
        Two c = new Two();
        Three d = new Three();
          d.print();
          c.print();
```

← d retain property of super class

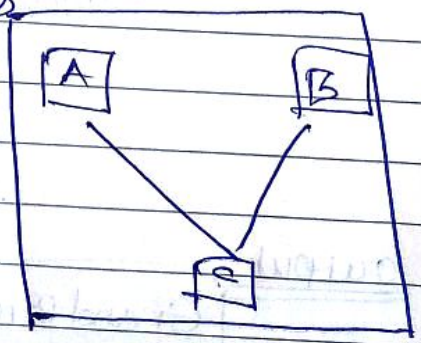
output

```
Shubham
Sawant
```

4) Multiple Inheritance (Through Interfaces)

Java don't support multiple inheritance through classes but allow through interfaces.

In which one subclass inherites many superclasses.




```
interface one {
    void Print1(); }
interface two {
    void print2(); }
interface three implements one, two {
    void print1();}
class child implements three {
    void print1() { sysout(
```

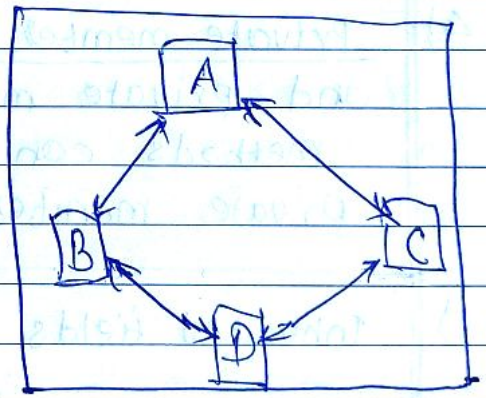
Q. why java don't support multiple inheritance? classmate Date Page

```
void print2(){ sysout("sawant") }
class Main{
    public static void main(String[] args){
        Child a = new child();
        a.print1;
        a.print2;
```

output

```
Shubham
sawant
```

5) Hybrid Inheritance (Through inheritance) (Only Interfaces)
mixing of two or many inheritances as multiple inheritance is not supported in java it's obvious Hybrid not supported by class but allowed for interfaces.



Q. Why java don't support multiple inheritance?
Because if A & B have same methods and variable this A,B extended by C then their will be runtime error because it's become ambiguose to call method. hence to avoid this ~~to~~ java don't support it. and throws compile time error which better than runtime error.

Important fact

1) Default superclass - Every class has one and only one superclass in absence of superclass class has superclass by default that is 'Object class'.
Exception - Object class don't have any superclass

2) Superclass is only one - As Java don't support multiple inheritance.

3) Inheritance of constructor - Subclass inherites all members (fields, methods & nested classes) but constructor not member hence not inherited. But they can be revoked by subclasses.

4) Private member Inheritance - Public, Protected member and private members through (getter & setter) methods can be accessed by subclass but private member cannot accessed directly.

5) Inherited fields and methods both can be used directly

6) We can have new methods & fields in subclass which aren't in superclass also we can override existing methods.

7) We can have new static methods that has same signature in superclass thus hiding it. Static method in Base class conceals method in subclass.

8) We can write subclass constructor that invokes that constructor of superclass implicitely or by keyword (super).

Method overriding

when we create same function in child as it is in Parents but Impliment differently then that method is said to be overridden.

Parent

```
class parent{
    Public void Print(){
        sysout("Parent)
    }}
```

child

```
class child extends Parent{
    public void Print(){
        Sysout ("child")
    }}
```

```
class main{
    Public static void main (string[] args){
        Child c = new child();
        C.Print()
```

Output

child

hence method in parent (Print()) is overider by method in child.

Static methods in Base & Derived -

Base

```
class Base{
    static void Print (){
        Sysout ("A")
    }}
```

```
class Derivered extends Base
    static void Print(){
        Syout ("B")
    }}
```

```
class main
    public static void main (String[] args){
        Base a = new Derived ();
        a. print();
```

output

A

In case of absence of static functions output would be B

Upcasting & downcasting

```
class Parent{
    void Print(){sysout("Teacher")
    }}
```

```
Class Child Extends Parent{
 void Print(){ syout("Child")
    }}
```

```
Class main{
    Public static void main(string[] args)
        Child c = new child();
        Parent a = a;
            // This is upcasting subclass instance is assigned to Superclass object.
            Its fine since child is derived from Parent.
        child c = new child();
        Animal a = c;
        Child c1 = (child) a;
            // This is a downcasting or explicit casting works fine because c is child.
```

Constructors & Inheritance

```
class Base{
    int x;
    Base(int _x){
        x = _x;
    }}
class Derived extends Base{
    int y;
    super(x);
    y = _y;
    void display(){ sysout("x = "+'x'+" y = "+y)
    }}
class main{ Public static void main(String[] args){
        Derived d = new Derived(10,20)
        d.display();
```

Super keyword → Inherited constructor variables.

Output: x=10 y=20

Abstraction

Q. Can there be a abstract method without abstract class?

Q. What is abstraction? What are the ways to achieve it?

Abstraction - It's a property by which imp. details are displayed to user and remaining are not displayed to user. e.g. Car viewed as car not as components.

ways of achieving abstraction

1) Abstract class

i) Abstract class have both abstract & concrete methods. (final & static methods also)

ii) Any class having abstract method must declared as abstract

iii) Object of Abstract classes are not produced with (new).

iv) Abstract class have default constructor, and can have parameterised contructors.

2) Interfaces - We can achieve abstraction by using interfaces also ~~and~~ i

Q. What is abstract class?

Q. Can we define abstract method as final?

Q. Is it possible to instantiate abstract class?

Abstract Methods

i) Abstract method is method declared without implementation (body)

ii) Abstract method always ~~write~~ redefined in subclasses and compulsorily overridden. or either make subclass as abstract. hence cannot be final.

iii) All abstract method must implement in extended subclass.

When to use?

i) when we want to define superclass that gives generalised form of methods and field without providing its actual implementation of every method and then subclasses are supposed to fill it in details.

ii) When two or more subclasses doing same things in different ways through different implementation.

Class in abstraction some observations.

1) Instance of abstract class cannot be created but we can give refference of abstract class.

2) Abstract class can have constructors and called when instances of inherited class created.

3) Abstract class without abstract method is allowed

4) Abstract class can have final methods also that works.

5) Abstract class cannot have static methods they won't allow method overriding which is important in abstract

6) Abstract class and Abstract methods cannot be made final as abstract suggest incompletion and need to be overriding but final excusively means complete it gives compile error.

7)

```
abstract Class Base {
        abstract void fun()
                    }
Class Derived extends Base {
        void fun() { sysout ("Derived class")
                    }

abstract Class Parent {
        abstract void method1()
                 void method2() { sysout("Parent method2")
                                }
        final    void method3() { sout("Parent method3")
                                }
                 Base() { sysout ("Parent constructor")
                        }
Class Child extends Parent {
        void method1() { sysout ("child method1") }
        Child() { sout ("child constructor") }

Class main {
    Public static void main (String[] args) {
```

output (child method1)

```
        Parent a = new child();
        a.method1();        // @ overriding given out is 1
```

output (Parent method3)

```
        Parent b = new child();
        b.method3();        // without overriding access final method through child class of parent.
```

output (Parent method2)

```
        Child c = new child();
        c.method2();        // access method of parent through child
```

output (Parent constructor)

```
        Child d = new Child(); // only creation of object calls constructor.
```

Q. difference between Abstraction & Encapsulation.

Abstraction	Encapsulation
Hiding unwanted details to reduce complexity and showing only imp. information.	Hiding for security of fields. Hiding of details and mechanisms how object does something.
focussing on what info. object must have	Binding code & data in single unit.
Abstraction solve issue at design level	Encapsulation solve it at implementation level
Implement by abstract class, abstract interfaces.	Implement by using access modifiers.
object that helps in abstraction are encapsulated	object that result in encaps need not to be abstracted
Use for gaining the information	Used for containing the information.

Q. Difference between abstract class and interface.

Abstract class	Interface
i) It has both abstract & non abstract methods	i) It has only abstract methods.
ii) Don't support multiple inheritance	ii) Supports multiple inheritance.
iii) It can implement interface	iii) It can't implement Abstract class
iv) It has protected method Public abstract methods.	iv) It has public abstract methods only.
v) It has/can have static, ~~Public~~ final, ~~Static~~ final; variables with any access specifier.	v) It has public static final variable.

Important Points

1) ~~static abstract~~ abstraction methods and fields cannot be static as in abstraction subclasses has to override abstract me but static won't let overriding hence it gives complite er

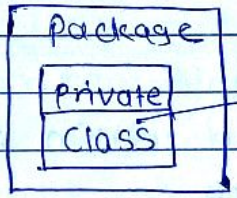
2) final cannot be used for abstract classes & abstract me whearas only final method can be used in abstract class.

Encapsulation

Encapsulation - Wrapping data and code together in single unit.
It is achieved by using Packages and Access modifier

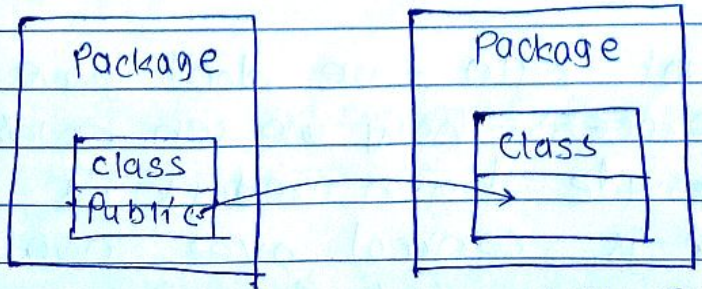
Access Modifiers

Private - Allows use of features within that class only cannot accessed by outside of class.



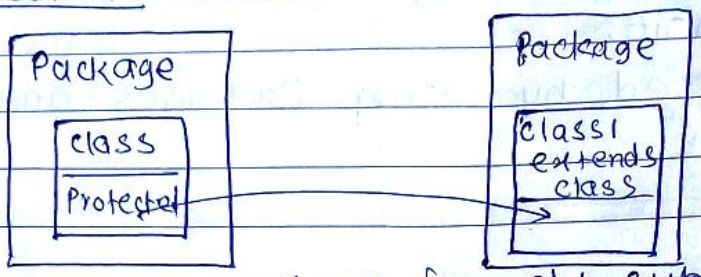
Don't allow use outside class

Public - Allows use of features by any class outside class and outside packages.



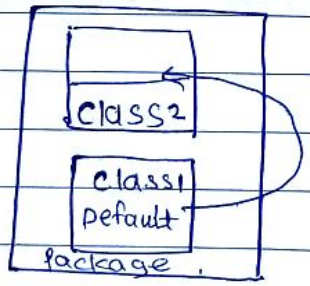
allows use outside package & outside class

Protected - Allows access for child classes only



allows for only subclass even if in another package.

Default - Allows use of feature within same package not accessible outside package.



allows for outside class within package.

Reallife e.g.

1) School bag is treated as encapsulation which stores books, notebooks, pen, stationary altogether.

2) Gmail account login - we don't have controll over the process "how we can enter we only can provide login credentials to login hence their is control over how we login we cannot interfere in process how credentials are treated

1) How to achieve Encapsulation

i) Declare variable of class as private.
ii) use getter & setter methods to write & read variables.

2) use of Encapsulation

It helps to control the modification of data fields.
It helps to decouple the components of system these decoupled components can be devloped tested and debugged independantly & concurrently any changes in one component dont have effect on others.

3) combination of data hiding & absraction.

Encapsulation make data of class hidden by 'private' by data hiding concept and expose the class to world without providing details about implemantation using concept of abstraction hence called as combination of data hiding & abstraction.

4) Example

```
Public Class Encapsulate{
        Private string StudentName;
        Private int StudentRoll;

Public getName() {
        return StudentName;
}
Public getRoll() {
        return StudentRoll;
}

Public void setName(String newName) {
        StudentName = newName;
}
Public void setRoll(int newRoll) {
        StudentRoll = newRoll;
}
```

get function used to access values of function used in objects.

set function used for setting the/assigning values of function and to be used by objects.

Q. advantages of encapsulation?
Q. How class can be made read only ~~write only~~?

5) Data hiding Vs Encapsulation

Encapsulation causes data hiding as we use private access modifier but Encapsulation itself is not a data hiding, it's more than that. as it also enable us to control modification of data.

Example continued ---

```
Public class TestEncapsulation{
    Public static void main (String[] args){
        Encapsulate obj = new Encapsulate();
        Obj.setName("Shubham");
        Obj.setRoll(33);
        Sysout(obj.getName());
        Sysout(obj.getRoll());
```

This function are public in Encapsulate class so can be access

set function used to assign values,

get function returns values as declared in its function

Output

```
Shubham
33
```

6) Advantages of Encapsulation

i) Data hiding - It is not visible for user how data is stored by class in variables, he only knows that passing value to setter method and getting initialized with that value.

ii) Increase functionality/flexibility - we can make variable of class read only or write only by using getter and setter functions. respectively.

iii) Reusability - Reusable program and easily change with new requirments.

iv) Testing code is easy - Easy to test for unit testing.

v) We get control over how data is modified.

Class

Class -1) Blue print from which objects are created

2) It has i) Data
ii) Methods
iii) constructors

3) It maintance hierarchy by inheritance property

4) maintain access specification of member variable.

Declaration

Access modifiers { Private, Protected, public, default } — Keyword class — Name in camelCase

```
Public   class   MyClass   {
```

Class body

```
    String  name;
    String  last name;
    int     roll no.;
    int     rank;
}
```

Q. what is objects?

Objects

Objects - Basic unit of OOP represents real life entities.

i) It has a) attributes - these are properties of object.
b) behaviour - these are methods or object.
c) Identity - that is name of object.

ii) They used to call non static function from class which are not present in main method.

iii) When object is created for class then it is called as instantiated class now attributes & behavi of class shared by different objects but value of those attributes are different for each object

Instantiation of object

```
class Dog{
        String name;
        int age;

    Dog(String name, int age){
            this.name = name;
            this.age = age;
            }

    }

public class Test {
        Public static void main(String[] arg){

        Dog dog1 = new Dog("Bravo", 5);
        Dog dog2 = new Dog("Oliver", 6);
```

Dog: class name; dog1 / dog2: object name; new: keyword; Dog: class name; ("Bravo", 5) / ("Oliver", 6): Parameter/Attributes to assigne to object

```
        Sysout(dog1.name + "_" + dog1.age);
        Sysout(dog2.name + "_" + dog2.age);
```

Output

Bravo 5
Oliver 6

methods to declare object

1) using new

```
ClassName  obj.reference = new ClassName (int a, int b)
```

- ClassName: class name
- obj.reference: name of object to be created
- new: new keyword
- ClassName: class name
- (int a, int b): Attributes parameters

Class	Object
i) Blueprint to create objects	i) Instances of class
ii) Similar objects stored in class	ii) Real world entity.
iii) logical entity	iii) Physical entity.
iv) class declared once	iv) declared many times in class.
v) Don't allocate memory	v) Allocates memory
vi) created by 'class' keyword	vi) Created by new keyword
vii) One way to create class ie class keyword	vii) many ways by using a) new b) new instance() c) clone() d) deserialization.

Q. what is interface?
Q. can interface method be static? can interface be final?

Interfaces

Interface - Its a collection of methods which are abstract also have methods like default, static which have method bodies and also nested type method.

Properties apart from class -

i) Interface cannot instantiated but they implemented on class and class is instantiated, all the methods in the interface must be implemented in class

ii) They don't have constructors

iii) All methods are abstract and must be implemented in class and they cannot be static as we cannot use static and abstract together.

iv) Fields should declared static and final but interface cannot be final as it will be implemented by other class

v) They Extends multiple interfaces.
also classes can implement multiple interfaces hence multiple inheritance achieved.

Example

Keyword to declare interface

```
interface Animal {
        public void eat();
        Public void travel(); }
```

Interface & methods in it are implicitly abstract we not need to define by abstract word

Implementing to class

```
Public class mammal implements Animal {
        public void eat () { sout ("mammal eats") }
        Public void travel () { sout "(mammal travel") }
        public int nodlegs() { return 0;}
```

We can have other methods in class →

main method

```
        mammal m = new mammal();
        m.eat();
        m.travel();
}}
```

output

mammal eat
mammal move

Members in interface are public by implicit- hence cannot declared as private, protected etc. also should be declared static & final

Methods are abstract hence cannot be static

They Don't have constructor.

Interfaces cannot final because they implement others.

Q. what is constructor
Q what are types of constructors

Constructors

Constructors - used to initialise the variables in classes

i) By default when object is created one constructor is invokened to assign values of variables this values by default are 0, null, ~~false~~ 0.00, 0 ~~whether there is no constr~~

ii) like methods constructor also has some statement that are executed once object created.

iii) Their is no return statement in constructor as it only returns current class instances. we can write 'return' in constructor.

```
Class MyClass {

        String name;
        int age;

public  MyClass (String name, int age) {
Access modifier  Construct name same as class name   parameters from which used in object creation.
        this.name = name;

        this.age = age;
```

this keyword used to give reference of variable in class body ourside constructor

```
}
```

iii) Constructor cannot be

a) abstract
b) final - As we just initialise value by constructor
c) static
d) Synchronized

Q. what is default constructor & if not provided any constructor what are the default values.

Argument constructor	Non Argument constructor
i) constructor with parameter	i) No parameters
ii) when constructor is created wheather it is argument or non argument compiler won't create default constructor.	ii) By default compiler create No arg. constructor

Overloading of constructor.

we can overload constructor as same as methods, com differentiate between this constructors on the basis o number of and type of parameters.

```
class MyClass {

    public Myclass (int a, int b) {
                                    }
    public Myclass (String a, int b){
                                    }
```

overloading

Default constructor

When constructor is not specified Java compiler provide default constructor in which default values of int is 0
String is Null

Whenever object created using 'new' keyword the default constructor is called.

Q can constructor be final?
Q can constructor inherited?
Q Can we overload constructor?
Q Can constructor return anything?
Q what is copy constructor in Java?

classmate
Date
Page

Return Type - constructor do return 'Yes' but constructor returns current instance of class (we can't explicitely return anything with constructor)

copy constructor - There is no copy constructor in Java as in C++ but we can use other methods like i) clone method ii) constructor iii) Assigning value of object to another to copy constructor data.

```
Class student{
    int id;
    String name;
    Student (int i, String n) {    // constructor
        id = i;
        name = n; }
    Student (student) {            // copying data to another constru
        id = s.id;
        name = s.name;
    }
```

In main method we made two objects S1 and S2 and these are constructors for those objects.

Constructor chaining -
constructor chaining is the process of calling one constructor from another constructor with respect to current object.
done in two ways -
i) From same class - (this)

ii) From Base class - (super)

Used when we

Q can constructor be final?
Q can constructor inherited?
Q can we overload constructor?
Q can constructor return anything?
Q what is copy constructor in Java?

classmate
Date
Page

Return Type - constructor do return 'Yes' but constructor returns current instance of class (we can't explicitely return anything with constructor)

copy constructor - there is no copy constructor in Java as in C++. but we can use other methods like i) clone method ii) constructor iii) Assigning value of object to another to copy constructor data.

```
Class student{
    int id;
    String name;
    Student (int i, String n) {    // constructor
        id = i;
        name = n; }
    Student (student) {            // copying data to another constru
        id = s.id;
        name = s.name;
    }
```

In main method we made two objects S1 and S2 and these are constructors for those objects.

Constructor chaining -
constructor chaining is the process of calling one constructor from another constructor with respect to current object
done in two ways -
i) From same class - (this)

ii) From Base class - (super)

Used when we

Q Difference in constructor & methods.

Difference in constructor & method

constructor	method
i) used to initialise state	i) used to define behaviour
ii) should not have 'return'	ii) should have return type
iii) If we don't provide compiler provide the default constructor	iii) Don't have default method
iv) name should be same as class name	iv) No restriction on name
v) implicit	vi) Explicit.

Destructor - Its a method called when destruction of an object takes place. "The main goal of destructor is to free up memory

Functions

1) close open files
2) close Database connections
3) close network resources.
4) and destroy object and free memory

Syntax

```
Class object{
    protected void finalize()
                { //Statements like closing database
                }
}
```

Working - When object is no longer in use it can be available for garbage collector to clear the heap memory but before doing that JRE calls finalize method closes all connection and destroy object

Advantages

1) Destroy value created by constructor
2) called at the end of program
3) Never overloaded as no arguments given in it.

e.g.

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

        Integer i = new Integer(2);
        i = null;
        System.gc();
        Sysout ("In the main method");
    }
    protected void finalise()
    {
        sysout ("object is garbage collected")
    }
}
```

Output:

In the main method

	Constructor	Destructor
i)	For Initialising instance of class	i) delete the object when not in use.
ii)	called when object created	ii) called when Object destroy
iii)	memory allocate	iii) memory Release
iv)	overloading Possible	iv) No overlodding
v)	Have Arguments	v) NO Arguments.

Methods

Methods - method is collection of statements that perform some specific task and return results in given data for

1) It can operate without returning result which is differ concept other than C++, Python
2) Saves time & efforts.
3) We can add Exception list in method statements.

```
Class MyClass {

    Access modifiers   Return type   method name start with small   Parame
    Public Static      Void          methodName(int a, int b) {
    (Optional)         (Return data type, Void in case no return)   (method signature)
        if (a > b) {
            return a;
        }
        else {
            return b;
        }
    }

    Public static void main (String[] arg) {
        MyClass obj1 = new MyClass()
        int a = obj1.methodName(10,5);
        Sysout(a)
    }
```

Output:

10

memory

a) method Implemented by stack, whenever method is called it create stack frame allocates mem and after execution stack fram deleted and stack pointer registers adjusted accordingly.

Q. what is method overloading? Is it possible with changing return type?
Q. types of method overloading?
Q. can we overload method by static
Q. can we overloade main method.

method overloading

we can Overload methods i.e. creating many methods having same name but different signature.

```
class MyClass {
    public void method (int a){
        Sysout (a);
    }
    public void method (String a){
        Sysout (a);
    }
}
```

Overloading — Same name; different parameter

// if we create object as obj1

```
MyClass obj1 = new MyClass()
int a = obj1.method (null);
Sysout (a);
}
}
```

output

error as we use null which is applicable to both parameters in both methods so compiler confused which method to executed.

* method overloading not possible by changing return type only. to avoid ambiguity. (How we could invoke method then)
* types — i) overloading by different no. of data
  ii) overloading by different data types
* We can overload main method also as many times we want
* We cannot overloade method by making them static i.e. i) static method(); ii) method(); } we can't do this.

Q. In How many ways string created?
Q. Why we use string literal method?

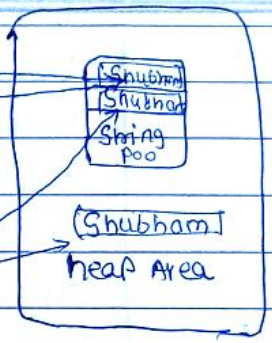
Strings

String is a object with array of char hence Strings are immutable as arrays and hence cannot grow and remains in memory forever, ~~only~~ whenever we make changes complete new String created.

String literal →
```
String a = "Shubham";
String s = "Shubham";
```
This method save memory.

This method create 's' at String Pool area also other reference given to same address every time new string created with same string

By New keyword →
```
String s = new String("Shubham");
```
This method creates two copy one at String pool and one at heap. but when their is copy already in string pool it won't create another



Hence both string are throwing false when compared as they are not equal due to different locations. But they can be compare as a same by using `equals( )` function

String is a sequence of characters

```
char a = 'A'
String a = "AA"
```

(+) operator can be used instead of concate) function.

```
String a = "Shubham";
    a = a + "_Sawant";
```

It gives a = Shubham Sawant.

Q. * If we don't provide any value in command line then value passed in string array in main method is empty not null.

Q. difference in string & string Buffer?

methods in String class

Char	charAt(int index)
int	length()
String	subString(int beginIndex)
String	subString(int beginIndex, int endIndex)
boolean	contains(char Sequence s)
boolean	Equals(another object) - Irrespective of storage only ~~comparison~~
boolean	isEmpty()
String	concat(String str) - adds str next to string object
String	replace(char old, char new)
String[]	split(String regex) - string regex is char around which spliting done and stored in array.
int	indexOf(char)
String	toLowerCase()
String	toUpperCase()
String	trim() - remove spaces before and after string.

	String	String Buffer
i)	Immutable	mutable
ii)	It create new instance on concating strings	It don't create new instance on concating.
iii)	No memory efficient	memory efficient.

String Buffer Class

String Buffer constructors

1) String Buffer ()
2) String Buffer (int size)
3) String Buffer (String str)

It allocates 16 character room
(int size) sets size of Buffer.

Methods

StringBuffer s = new StringBuffer ("Geeks for Geeks");

1) int a = s.length()
   13

2) int a = s.Capacity()
   29

3) append (String str)
   append (int num)

   s.append ("ss"); ⇒ GeeksforGeeksss
   s.append (1); ⇒ GeeksforGeeks1

4) insert (int index, String str)
   insert (int index, char ch)
   insert (int index, int a)
   s.insert (3, H) ⇒ GeeHGs for Geeks

5) reverse()
   s.reverse() ⇒ skeeGrofsKeeG

6) delete (int start index, int end index)
delete charAt (int location)
S. delete charAt (5) ⇒ Geeks or Geeks

7) replace (int start index, int end index, str)
S. replace (5,8,"are") ⇒ Geeks are Geeks

8) eSure Capacity (int Capacity)
It sets capacity larger of a) specified by int capacity
b) twice capacity + 2

9) CharAt (int index)
S. charAt (5) ⇒ 0

10) int Stream chars( ) - returns stream of int zero extending char from this sequence.
S. chars( )

11) int codePointAt (int index) - returns character at given index
S. codPoint At (int index)

12) int codePoint Before (int index) - returns character (unicode code point) at before given index.
int Stream codePoint( )

13) int indexOf (String str) } search index of first occurance of element
indexof (String str, int fromIndex) }
lastIndexOf (string str) } search last occurance of element
lastIndexOf (String str, int from index) }

14) void setCharAt (int index, char ch)
Character at specified index set to ch.

S. difference in StringBuffer & Builder

void setLength(int newlength) - sets new length of String.

CharSequence subSequence(int start, int end) - returns new char seq.
S. subSequence(5,7) ⇒ ~~Geek~~ for

String subString(int start) - } returns new sub string
subString(int start, int end) - }

String toString() - this method returns new string representing data in this sequence.

void ~~t~~trimToSize() - this method used for reducing the capacity of String to save space.

String Builder

* String Builder provide same methods as that of String Buffer only differs in synchronization
* String Builder has no guarantee of Synchronization.
* String Builder is used for single thread and gives faster implementations & more efficient.
* String Buffer used for multithreads. hence recommended.

String Tokenizer class

Constructors

StringTokenizer (String str)
default delimiters are spaces, tabs, new lines, carriage return

StringTokenizer (String str, String delim)
given delimiters as string.

StringTokenizer (String str, String delim, boolean flag)
if flag is false Gives tokens as "Geeks", "Hello" and "_" is delimiter.
if flag is True Gives tokens as "Geeks", "_", "Hello" and "_" delimiter.

```
import java.util.*;
public class newClass
{
    public static void main (String[] arg){
StringTokenizer S = new StringTokenizer ("How:are:you",":");
    while (S.hasMoreTokens())
        sysout (S.nextToken());
```

Output

How
:
are
:
you

Methods in StringTokenizer class

1. hasMoreTokens() - used when boolean value required useful in while loop as earlier.
   - True if next token exist (next to current token)
   - else false.
2. nextToken() - return next token
3. countToken() - returns total no. of tokens
4. nextElement() - } Used as same as hasMoreTokens() & nextToken() but is usefull in Enumeration Interface. C
5. hasMoreElements() - }

Array

Array - Its a data structure which stores fixed size sequential collection of elements of the same type not of diff. type as in python.

Syntax

declaring array

Allocating to memory of array.

```
datatype [] arrayrefvar
arrayrefvar = new datatyp [arraysize]
```

OR

```
datatype [] arrayrefvar = new datatype [size]
```

e.g.

```
int [] a = new int [5];
```

Important points

1) JDK 1.5 introduced new loop i.e. for each loop

for each loop

```
double [] a = {1.9, 2.1, 3.2, 4.5};
    for (double element : a) {
        sysout(element);
```

hence we can use loop to access array elements.

2) we can determine array length by arrayname.length(syntax.

```
Int [] a = new int [5];
x = a.length;
```

x gives array length i.e. 5

3) We can use int value & double value but short & long not use

4) we can used Array as Static field, local variable & method parameters.

5) By Default values are Null, false or zero according to type of data

6) when we try to access array element out of array size JVM throws ArrayIndex out of bound Exceptions.

7) we can pass array to method as arguments

8) we can return array from methods.

9) Arrays are object of class and direct superclass of arr is class object. ~~the~~ all methods are inherited from class object only clone method is not inherited for multidimensional array shallow copy created

Advantages

i) we can access any element easily by index.
ii) can store many element at a time
iii) Primitive type to wrapper classes object conversion will not happen so it is fast
iv) cashe locality means elements in array stored in nearby places in memory so improve performance unlike of linked list.

Disdvantages

i) Store only single type of data
ii) don't have add or remove method
iii) Fixed length & have to mention which can cause wasted memory.
iv) To delete element have to traverse through whole array

Multidemensional Array - These are arrays of arrays with each element of array hold reference of other array also known as Jagged array.

Syntax

DataType [][]arrayrefvar = new Datatype [size][size]
DataType [][][] arrayrefvar = new Datatype [size][size][size]

OR

Datatype [][] = {{x,y,z},{x,y,z},{x,y,z},{x y z}}

Volatile Array

Location of storage

Array class In Java.

class hierarchy

java.lang.object
  java.util.Arrays.

```
int[] x = {1,2,3,4,5,6,7,8,9,10};
int y = 2;
int[] x1 = {1,2,3,4,5,6,7,8}
Arrays.sort(x);
```

Methods in java.util.Arrays.

1) Arrays.asList(x) - returns fixed size list back by specified

2) Arrays.binarySearch(x,y) - Return index of element y by binary search algorithm.

3) Arrays.binarySearch(x,1,3,y) - search from 1 to 3 indexes.

4) Arrays.copyOf(x,20) - copy array and size is set to 20.

5) Arrays.copyOfRange(x,1,3) = copy elements from index 1 to index 3 and create new array.

6) Arrays.equals(x,x1) - Return boolean by comparison

6) Arrays.deepEquals(x,x1) - Return boolean by deep comparing deeply both arrays.

7) Arrays.deepToString(x) - Returns string representation of deep contents i.e. multi dimensional.

8) Arrays.fill(x,1,5,10) - Returns new array with value to filled at index 1 to index5.

9) Arrays.hashCode(x) - Returns hashcode.

10) Arrays.mismatch(x,x1) - Returs index where first mismatch element found in both arrays.

11) Arrays.setAll(x, functional generator) - sets elements of array using generator function. provide

12) Arrays.sort(x) - set in ascending order.

12) Arrays.sort(x, 1, 5) - sets elements from 1 to 5 index in ascending order.

12) Arrays.sort(x, collection.reverseorder()) - descending.

13) Arrays.toString(x) - Returns string of array element covered with '[' and ']' by either sides

Object class in Java

Object class present in Java.lang package Every class in Java directly indirectly derived from Object class root of inheritance hierarchy.

Usefull Classes In Java

GregorianCalendar

Date

Camelcase letters
e.g "HelloWorld" no space & Alphabets are capital.

Errors
1) compile time - which is given by IDE itself.
2) Runtime time Error while running code.

Difference in C++ Java

Tail Recursion
multiple inheritance.

Memory Management

"Memory is managed in Java" - As unlike other languages in Java developer has no need to explicitely allocate or dellocate memory as this is done by JVM and GC in JAVA hence memory is managed in JAVA.

STOP the World - when GC thread running other threads are stopped means application stopped momentarily this causes unacceptable freeze hence its a disadvantage of JAVA GC and we need to be sure for least freezing should be encountered by tuning the GC and optimizing JVM.

Memory Management.

"Memory is managed in Java" - As unlike other languages in Java developer has no need to explicitely allocate or dellocate memory as this is done by JVM and GC in JAVA hence memory is managed in JAVA.

STOP the World - when GC thread running other threads are stopped means application stopped momentarily this causes unacceptable freeze hence its a diadvantage of JAVA GC and we need to be sure for least freezing should be encountered by tuning the GC and optimizing JVM.

Memory In Java

Heap ~~Method Area~~

- It is fixed or dynamic hence can be controlled
- It stores actuall objects created, Arrays by 'new' keyword.
- created when JVM starts, only one heap.
- when heap is full garbage is collected

Divided further into following structure

i) Young Generation
   - Eden space
   - S0 survival space
   - S1 survival space

minor GC are performed on new objects stored in Young generation and survived objects after Reaching certain age moved to old generation.

ii) Old Generation (Tenured Space)
Survived objects from minor GC in Young generations moved to old generation to perform major GC.

iii) Permanant Generation
Used to store metadata of JVM to describe class, methods stores runtime based classes.

iv) code cache.

method Area

- Part of Heap area
- creates when JVM starts
- Stores class structure, constructors, superclass, Interfaces, datatypes like String, modifiers, names of superclasses, interfaces
- It may or may not garbage collected. as it is compulsory in heap

Q How many types of memory areas are allocated by JVM?

JVM Stacks

- It can be fixed or dynamic hence sizeof stack can be choosen independently when created.
- Stores data and partial results like returns of methods, ~~dynamic~~ linking, ~~Exceptions~~ dispatch.
- Stores Refferences of objects stored in Heap.

Native method Stack (C) stacks

- It can be controlled, can be fixed or dynamic
- Stack for native code written in other than Java languages
- Java Native Interface (JNI) calls it.

PC Registers
(Program Controller)

- Each thread has (PC) for it
- Pc Register stores Return addresses and native pointer and address of jvm instructions

Q How garbage collection controlled?
Q. what is garbage collector?
Q. what is gc()?

classmate
Date
Page

Garbage Collector

Overview - It is used for freeing up memory form heap area by deleting unreachable objects only functional for heap area.

Syntax for calling the GC are
runtime.gc() and system.gc()

But although it can be called explicitly final decision of GC is depends on JVM. JVM Runs GC if it sensess memory running low.

- Object is considered for GC if it is unreachable when no live thread can access it; if object has variable reference and that reference is available for live thead then object is Reachable

Types of GC

Serial GC - uses mark and sweep approach for young and old generations

Parallel GC - It spawns N (cores in CPU) threads for young generation garbage collection. similar to serial.

Parallel old GC - similar to parallel but multiple threads for bothe generation

Q. How garbage collection is controlled?

Concurrent mark sweep (GL) -

GL (GC) - Its a Replacement for CMS collector, Parall concurrent, incrementary compacting low pause garbage collector.

When we call GC by System.gc() on the object inside main method, the it calls the finalize() method from class whose object is created.

```
public class test {
    public static void main (String[] args){
        String str = new String ("shubham");
        str = null;
        System.gc();
        Thread.sleep (1000);
        sout ("end of main");
    }
    protect void finalise(){
        sout ("finalise called");
    }
}
```

output

end of main

As finalise called from its class string and not on the test class.

Q. How object can be unreferenced?

gc() - method to invoke garbage collector's cleaning process. This method make free up space occupied by unused objects by JVM explicitely.

Unreferencing

1) By Nulling the reference.

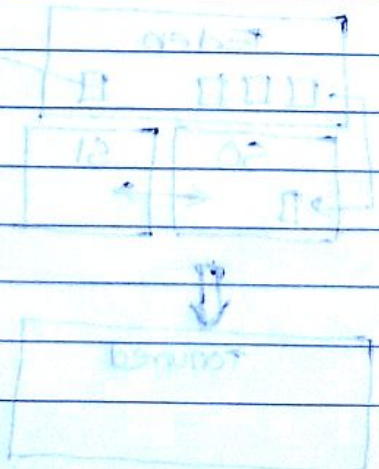
```
Employee e = new Employee();
e = null;
```

2) By assigning reference to another

```
Employee e1 = new Employee();
Employee e2 = new Employee();
e1 = e2;
```

3) By anonymous object.

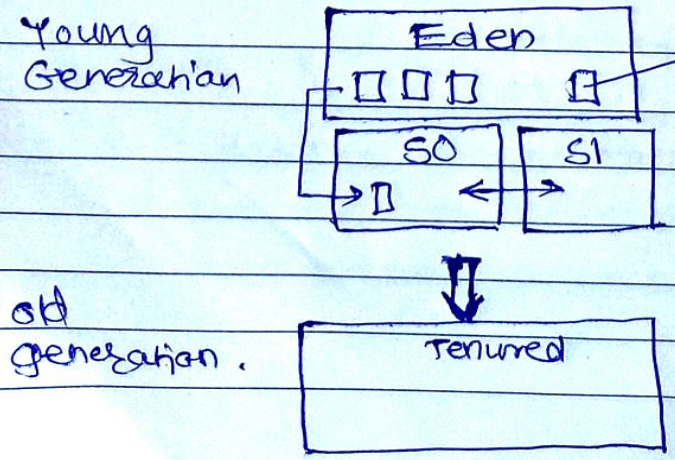
```
new Employee();
```



Generational Garbage collection

Steps

1) Firstly new object allocated in Eden
2) when Eden space is full minor gc performed and referenced objects moved to Survival S0 Space and unrefferenced deleted. and eden space cleared for new objects.
3) On next minor GC Eden and S0 cleared and object stored to S1 hence S1 has both aged objects. (aged objects from S0 & new from Eden). Referrenced objects of Eden and survived aged of from S0 moved to S1.
4) On next GC same thing happens only Survival Spaces changed, (Eden) and (S1) cleared and (S0) survived objects are aged.
5) when object reaches certain age they moved old generation spaces.
6) Finally major collection of Garbage done to clear and compact space in old generation.



Q. What is exception handling? Hierarchy of exceptions
Q. types of exception occured

Exceptions

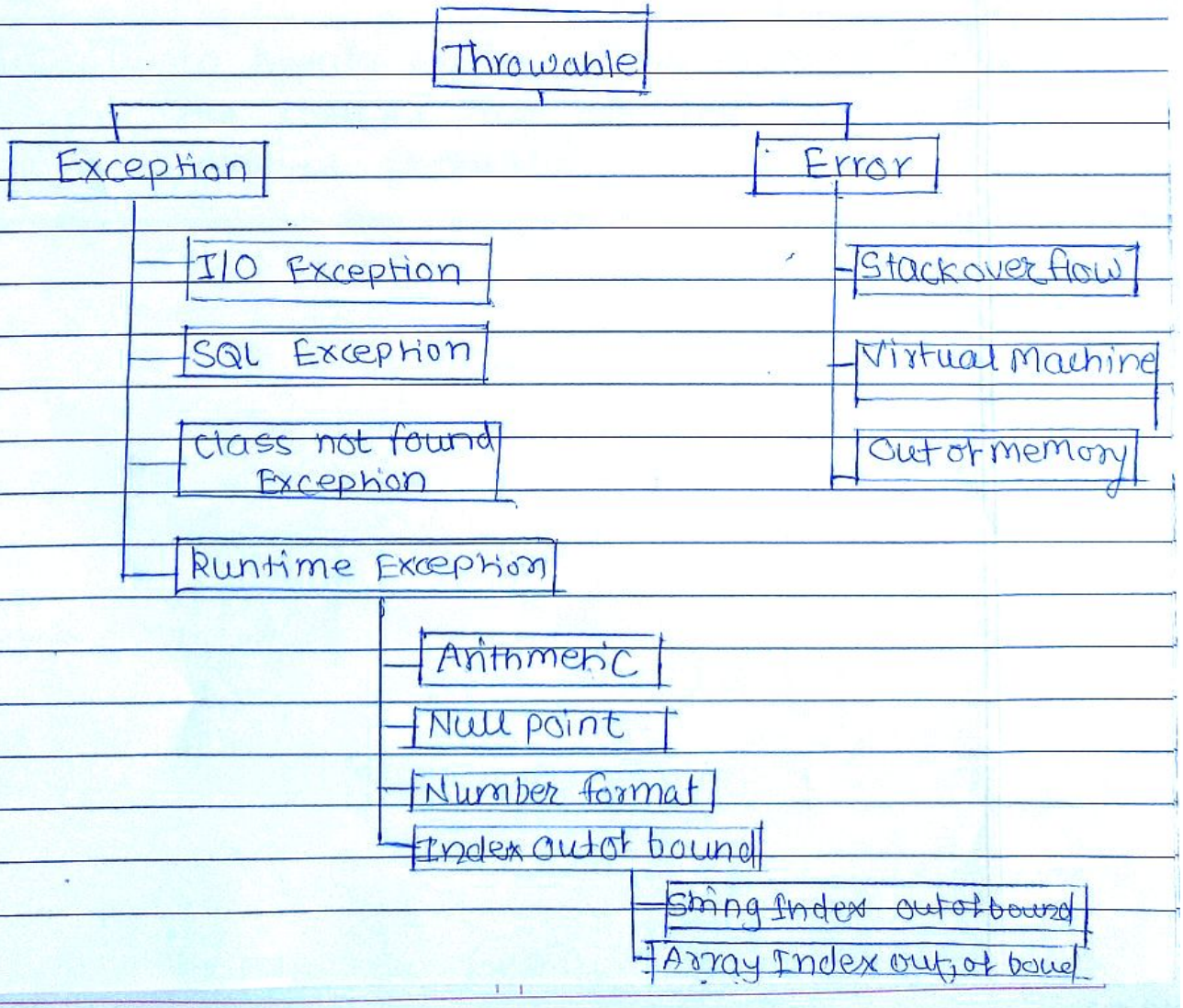
Exception handling - mechanism to handle runtime errors so that normal flow of application is possible

Illustration

```
Statement 1
Statement 2
statement 3   // Exception occurs
Statement 4
Statement 5
```

Exception handling allows us to proceed for next statements after 3rd getting exception, hence flow of application is not disrupted.

Hierarchy of JAVA EH (Exception class)



Q. what is base class for errors & exceptions
⇒ Throwable.

Common Scenario's

1) `int a=50/0` // Arithmetic Exception

2) `String s=null;` // Null point Exception
`sout(s)`

3) ~~String s="abc";~~
~~Int i = Integer.parseInt(s);~~ // Number format Exception

4) `int a[] = new int[5]` // array out of bound exception
`a[10] = 50;`

Q. difference in throw & throws.

Types of E	Keywords
try	to specify block of code where exception may occur and block followed by catch or final.
catch	to handle exception occured in try block we can handle it implicitely or explicitely
finally	finally is block of code which executes wheather exception occured or not, it is followed after catch keyword
throw	used to explicitely throw an exception custom exception.
throws	to specify that their may be exception inside the method and declared alway after method signature

Q. types of exceptions
Q. difference in checked, unchecked

Types of Exceptions

	Checked	Unchecked	Error
i)	The class which directly inherites 'throwable' class without 'Runtime' & 'Error' are checked Exceptions	ii) The class which inherites 'Runtime' Exception is unchecked Exceptions	i) Errors are non recov
ii)	Checked at compile time	ii) Checked at Runtime	ii)
iii)	e.g. I/o, SQL etc	iii) Eg. Arithmetic, Array out of bound etc.	iii) Out of memory Virtual machi Assertion Er

Q. is it necessary that every try block followed by catch block? ⇒ no it can be followed by finally block too.

Try-catch keywords

```
Public class Exception{
    public static void main(String[] args){
        try {
            int a = new int[5];
            a[5] = 30/0;        // first exception
            sout(a[10]);        // second exception
        }
        catch (ArithmeticException e) { sout("Arithmetic"); }
        catch (ArrayIndexOutOfBoundsException e) { sout("Arra
        catch (Exception e){ sout("Parent Exception"); }
        finally { sout ("finally block always executed"); }
        sout ("Rest of code");   // Rest of code will not executed without handling the exceptions
    }
}
```

Sequence of Exceptions catch should maintained General ⇒ specific hence (Exception) at bottom otherwise error thrown.

output

```
Arithmetic
finally block always executed
Rest of the code
```

// code has two exceptions but at a time only one exception is catched and invoked. sequencially from from top

'finally' block always executed even if their is Exception occurs or not it is catched or not.

8. what is finally block?

Try-Catch-finally keywords

Throw

```
int age = 13;
if (age<18){
    throw new ArithmeticException("not valid");
        }
else { sout("welcome to vote")};
sout("rest of code");
```

Exception in thread main java.lang.ArithmeticException
not valid

Hence throw is used to throw Exceptions explicit

Throws

```
class Exception{
    Void method1() throws IOException{deviceer
        throws new IOException("device error");}
    Void method2() throws IOException{
        method1(); }
    Void method3() throws IOException{
     try {method2();
         }
     Catch (Exception e){ sout(" exception handle
    }
public static void main (String[] args){
        Exception a = new Exception();
        a.method3();
        Syout(" normal flow");
}}
```

Output

```
Exception handled
normal flow
```

By throws keyword we can declare Exception

C	JAVA
1) Not OOP	2) OOP
2) Procedural	2) Non procedural but oop.
3) 32 keywords	3) 50 keywords
4) Default members are public	4) Default members are private
5) Data hiding done by static	5) Data hiding done by private
6) Used for application & system programming	7) Used for web application mostly.

	Java	C++
1)	compiled into bytecode then platform independent	platform dependent
2)	Java source code is byte code	program in C++ is compiled into object code
3)	Java translate code into byte code and hence portable for any platform	C++ code not portable it must compiled for each platform
4)	memory management is automatic with the help of JVM and GC	memory management is manual allocation/ Dellocation of memory done manually.
5)	only methods overloaded operator overloading not allowed.	Method and operators can be overloaded this is static poly morphism.
6)	Multiple inheritance only through interfaces	single and multiple inheritance is possible
7)	Java is using pointers less	C++ is all about pointers.
8)	Java have (/**---*/) this kind of documentation comments	C++ does not have documentation comments.

8. Difference in c++ and Java

	Java	C++
9)	Object oriented and have single root hierarchy	procedural & object oriented no single root hierarchy
10)	Source code & filename is same, easy to search.	source code filename don't have any relationship
11)	Weak encapsulation.	Strong encapsulation with access specifiers control access
12)	Java doesn't support 'Go to' statement.	C++ supports 'Go to' statements.
13)	Runtime error detected by System	Runtime error detection is responsibility of program
14)	Java don't support structure and union.	C++ supports structure & unions.
15)	Java supports only pass by value parameter	C++ supports pass by parameter pass by value
16)	Not close to hardware and close to web application.	Nearer to hardware because of many libraries it supports