

Working Of Garbage Collection.

- = 1) Garbage Collection is nothing but execution of `System.gc()` statement.
- 2) This statement can be called by either programmer or JVM but decision to perform garbage collection is taken only by JVM.
- 3) This means that although the programme would write `System.gc()` in his program there is no guarantee that JVM would accept the request to do so.

```
Ex: class A {
    public void psum() {
        System.out.println("Hello");
    }
}
```

At `a = new A();`

`A a1 = new A();` if a1 is not used

`a = null;` then all objects

`System.gc();`

}

Q: How many objects are present in heap.

Ans: Cannot be determined.

Packages & Access Specifiers.

Rules:

- 1) There can be only one public class per source code file.
- 2) If there is a public class in a file, the name of the file must match the name of the public class. for example, all classes declared as public class A must be in a source code file named A.java.
- 3) If the class is a part of package, the package statement must be the first line in the source code file, before any import statement that may be present.
- 4) If there are import statements, they must go between the package statement (if there is one) and the class declaration.

- 5) A file can have more than one non-public class.

Access Specifiers (Visibility Labels)

Def: μ ν obtain address A —
using label μ sends self

- Java Access Specifiers (also known as Visibility Labels) regulate access to classes, field and methods in Java. These specifiers determine whether a field or method in a class can be used by another class or sub-class. Access Specifiers can be used to restrict access.

Access Specifiers are an integral part of object-oriented programming.

Types of Access Specifiers.

Public
default
protected
Private;

Summary of access Specifiers.

Access Modifiers Def Pri Protected Public

- Accessible inside the class Y Y Y Y Y

- Accessible within Subclasses inside the same package? Y N N Y Y

- Accessible outside the package N N N Y

- Accessible within its Subclasses outside the same package N N Y Y

Primitive Data Type and Wrapper Classes.

Following data types are available in Java.

- i) Byte byte (8 bits)
- ii) Short (16 bits)
- iii) char (16 bits)
- iv) int (32 bits)
- v) float (32 bits)
- vi) long (64 bits)
- vii) double (64 bits)
- viii) boolean (true, false)

Ex: 1

int x = 5;

memory



range: -2^{16} to $+2^{16}$

Note:

$$2^{16} = 65536$$

$$2^{32} = \dots$$

it = b

word) requires just 8 bits of memory

ex 2: non explicit data promotion
byte b = 120; ✓

byte b1 = 130; X

$$2^8 = 256$$

range: -128 to +127

Eg: long l = 5l; ✓

①

int x = 6;

l = x; ✓

x = l; X

② float f = 3.2f;

double d = 5.5;

f = d; X

d = f; ✓

(3)

3 sides & 2 diagonals of \triangle are ml
 want east si (twice) fo
 float $y = 5.5$ ft; bus prob
 bottom of bus fo two parallel
 $y = \frac{x}{R}$; $\sqrt{b^2 + h^2}$ of si

$\sin(50^\circ) = \frac{h}{\sqrt{b^2 + h^2}}$ of \triangle are ml
 already a si at present
 in bus when total is added
 adding bus

(4)

float $f = 3.2 f_i$
 because si has no \triangle are ml
 on long ladder \Rightarrow half of
 bus missing at time t
 $d = f$; ~~\times~~ $f_i = 1.02$

not truly 3.2×2.3 \triangle are ml
 and that above if
 is not true \Rightarrow missing with
Note: boundaries start in

JVM says

~~Note~~

In eg ① as container size of $x(\text{int})$ is less than long and we are not losing out of any information it is allowed.

In eg ② $f = d$ means trying to fit a double inside a float which is not possible.

In eg ③ an int is assigned to float which means we gained the precision and so its fine.

In eg ④ although float can fit inside long we lose the precision and hence its not allowed.

~~ANSWER~~

P C
primitive casting.. old way (e)
conversion & cell (db) good

Eg:

① ~~long l = 5L; R (int) = l; print~~ ②

~~int x = 6; z = (x)~~ (paral)

~~l = x; f 2 11. 1 (2) good~~

① long l = 6; ✓

X: 001 → result of R (001 = 0.1) result ④
② long l = 6.0; X (Error: possible loss of
precision).

long l = (long) 6.0; ✓

sop(l); OLP → 6

③ float a = 5;
sop(a); // 5.0 ✓

④ double d = 5L;

sop(d); // 5.0 ✓

⑤ double d = 15.5f; ✓ assignment
sop (d); // 15.5 ✓

⑥ long l = (int) 5.5; ✓ cast (l)

(long) 5.5; ✓ cast

sop (l); // 5 ✓ assignment

✓ l = 2 eval / 10

⑦ char c = 'a'; ✓ char x = -180; X
char c = 100; ✓ char x = (char) -100;

✓ (0.2 (eval)) char y = -70000;
sop (c); // d

char y = (char) -70000;

✓ (0.2 (eval)) a = b 100 ✓ ⑧
✓ (0.2 (eval)) (a) goes

a = b 100 ✓ ⑨

✓ (0.2 (eval)) (a) goes

Widening

Widening and boxing in Java.

Not based on this note Priority

first of boxed & float

① Widening

second of boxed & int

② Boxing

third of boxed & long

③ Varargs

Ex: ① double d = 5.5f;

② float f = 5;

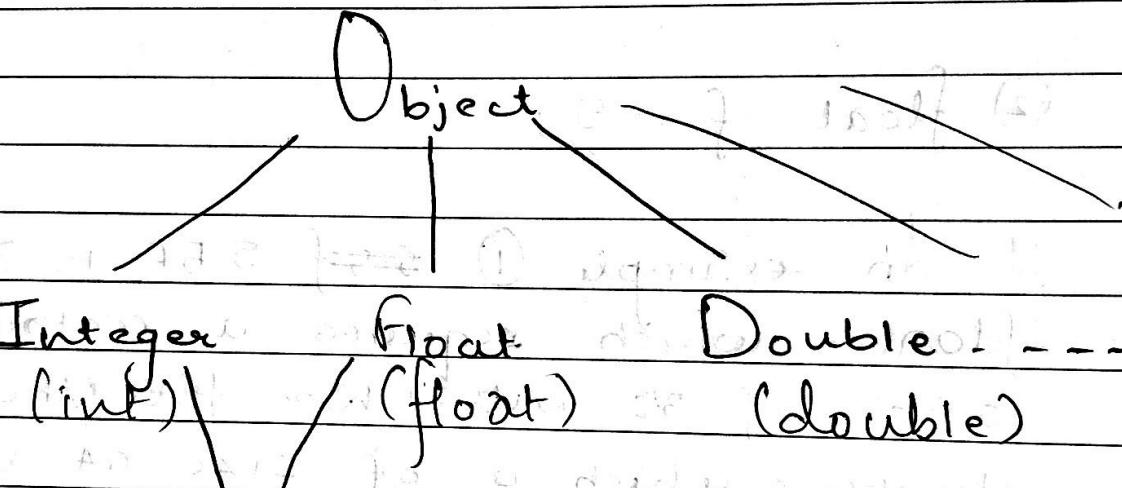
In example ① ~~if~~ 5.5f is a float which requires a container size of 32 bits. We fit this into double which is of size 64 bits.

This is widening from 32 to 64 bits.

In ex ② 5 is an ~~int~~ value which is assign to float and so becomes 5.0, this is also a second type of widening.

Boxing (Auto)

In Java, int is boxed to Integer, float is boxed to Float and likewise all primitive variables are boxed to respective classes.



(classes not related)

→ Q3) Variable of Arguments (Var-args)

No. of args depends upon

Ex: output without the first

① class A

```
sparks { blonde brown hair color } (2)
```

```
        brown void m1(int... x) test sd  
        { sout ("int"); }
```

X (P... 3rd > x[3]) in bio

↑
psvm(-)

.ps2 {i test}

new A().m1(5);

X (P new A().m1()) in bio

new A().m1(5, 5);

new A().m1(5, 5, 5, 5);

Generalization 3 to 4

3

Rules of 2 types of variables in C++

Format 1: int type in test

→ Q4) Int and Integer are similar in

variable arguments and hence

void m1(Integer... z) {} and

Void m1(int... y) {} are together not allowed.

② This also applies to float, Float;
Double, double and so on
for all primitive types.

(2) Variable argument should always be last (in the seq. of argument).

Ex: void m1(int x; float ... y) { } ✓

last in Seq..
↑

void m1(float .. x, int y) { } X

↑
Not attained.

(3) As variable args.. should be last in seq, we cannot write multiple arguments, bcoz both can't be last.

Ex (s...repd) for biov
(p...ai) for biov
• biov for

makes you happy?

Example:

Q10. ~~else~~ ~~int~~ ~~float~~

class A

(x3) for (z) in (x - int) for (7)

void m1 (int... z) // var-args

{

sout ("int ...");

}

(2 = 2 float) (x - int) for (5)

return void m1 (Integer... z) // Boxing

{

sout ("z Integer");

}

void m1 (float z) // Widening

{

sout ("z float");

}

{

new A () . m1 (5);

O/P float.

(x - float) / m

ntd's

calls

O/P

① m1(int.. x)

m1(5)

m2(object x)

m1(object x)

② m1(int x); short s=s;

m1(byte b)

m1(Short s)

③ m1(float x) m1(s); m1(long),

m1(long x)

m1(double x)

double

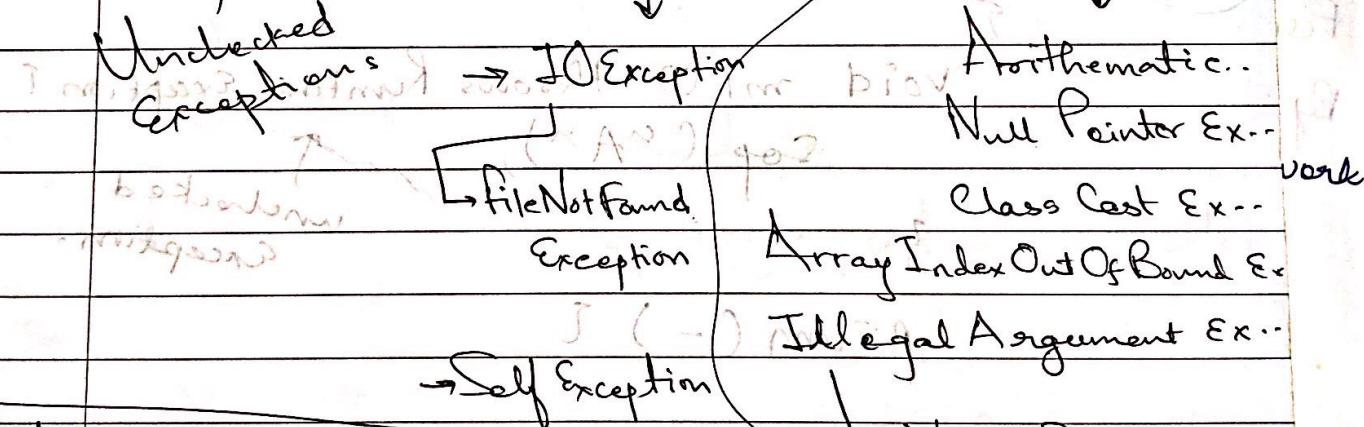
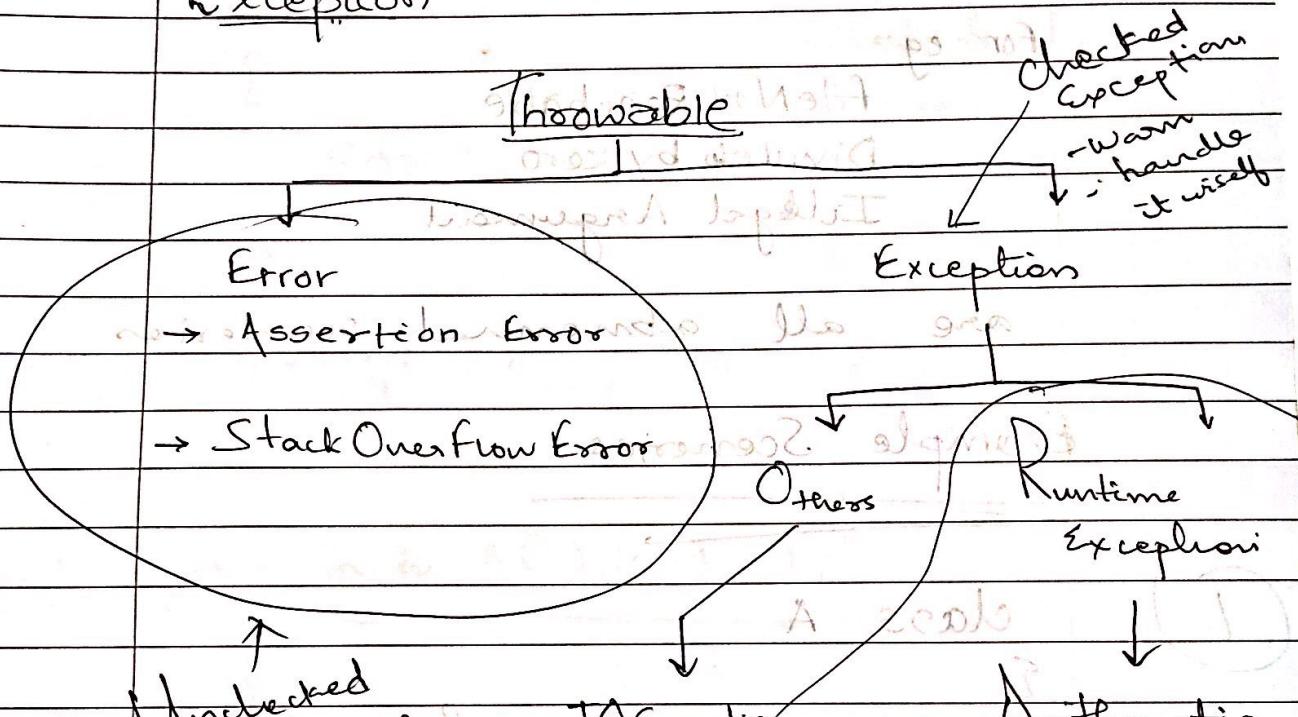
④ m1(long) m1(s, SF) m1(double)

m1(double n)

Exception Handling :

Why exceptions? in our programs
to handle errors

Exception



Unchecked exception.
(Do not check)

Unchecked
Exceptions

Exceptions are used in Java to warn or handle abnormal scenarios.

For eg:

FileNotFoundException

Divide by zero

Illegal Argument

are all abnormal scenarios.

Example → Scenarios of warning

① class A

void m1() throws Runtime Exception {

 System.out.println("in m1");

 ↑
 unchecked

exception.

 System.out.println("out m1");

 new A().m1();

(Abnormal flow of flow)

}

}

No need to handle or warn.

② class A { data: int }

The void m1() throws IOException

{ checked

Sop ("A"); b } below

3 below

psum()

- part (8)

is part (8)

{ below

(address)new AC). m1();

3 below

Need to

① try-catch (handle)

{ 3 part (8)

② throws (warn)

below

data part (8)

Try-Catch

Rules

```
1) try {  
    catch (Exception e) {  
        finally {  
            valid  
        }  
    }  
}
```

2) try { }
 "
finally []
 "
 catch (Exception e)
 {
 }
 }
 "
 valid

4) try { }
try { }
Catch (Exception e) { }

becoz 1 try - 1 catch.

Note. 1

There are 3 ways of throwing an exception. ~~so it's polymorphism of exception, for MVT is not~~

i) Write code that throws an exception

int a = 5;

int b = 0;

int c = a/b;

ii) Throw new ArithmeticException();

→ Manual throw.

iii) ArithmeticException e = new ArithmeticException();

throw e;

→ Manual throw.

Note 2.

To throw a checked exception manually, we have to warn the JVM first, however to throw unchecked exception there is no need to warn.

Ex:

① psvm() throws exception

{
 try {
 // some code
 }
 catch (Exception e) {
 e.printStackTrace();
 }

 try {
 // some code
 }
 catch (Exception e) {
 e.printStackTrace();
 }

 finally {
 // some code
 }

 finally {
 // some code
 }

(throws exception)
Manual Throw

② psvm()

{
 try {
 // some code
 }

 throw new NullPointerException();
}

}
 unchecked exc.

Example:

① class A {

```
    void main() {  
        int a = 10, b = 0;  
        try {  
            cout << "Hello" << endl;  
            cout << a / b << endl;  
        } catch (...) {  
            cout << "Arithmetic Exception" << endl;  
        }  
    }  
}
```

throw new ArithmeticException();
// Unchecked So no need to
3 warn or catch.

catch (ArithmeticException e) {

cout << "Arithmetic Problem" ,

} // "exception" two

finally {

cout << "Finally" ;

} // And so what

Output: { (2) (3) (4) (5) (6) (7) }

Output:

{ Welcome } two { please }

Arithmetic Prob .

Finally

Note:

If an unchecked exception is thrown and not caught,
we get a red message in the output. (Exception Thread Main)

② class A {

public void psvm() {

 try {

 System.out.println("Hello World");

 } catch (IOException e) {

 throw new Except

 } catch (IOException e) {

 System.out.println("Caught checked so either warn or
 catch or both");

}

 catch (IOException e) { // catching

 System.out.println("Arithmetic Exception");

}

 finally { System.out.println("Finally"); }

}

What makes you happy?

#HappyCollegeDays

Note: ~~checked (also) - slightly~~

If a checked exception is thrown,
We have to either catch it,

or ~~ignore it~~ or handle it

both

and we have to avoid catching and

(return of normal) presentation of exception

(throw normal exception) to application

(checked exception)

else

we can't catch the checked exception

and will return unhandled

(checked exception) to application

(checked exception) noted

Multiple-Catch Blocks

Rules

- a) In multiple catch blocks JVM selects most relevant first.
- b) We have to follow the hierarchy (lower to higher)
(means lower first)

Example.

```
try
{
    throw new ArithmeticException();
}
catch (ArithmeticException e)
{
    System.out.println("Arithmetic");
}
catch (RuntimeException e)
{
    System.out.println("Runtime");
}
catch (Exception e)
{
}
```

* ~~Exception in Overriden Methods.~~

3 Scenarios

1) void m1() throws Exception.

void m1() throws _____

1) All Unchecked

2) Exception

3) IO Exception

2) void m1() throws FileNotFoundException

(except throwable)

3) void m1() throws _____

void m1() throws _____

1) All Unchecked

2) File Not Found.

3) void m1() throws IOException, Error.

void m1() throws _____

1) All Unchecked

2) IO Exception.

3) FileNotFoundException.

* Self Exception:

- 1) To create our own exception class, it must extend generic exception class.

Ex:-

Class MyException extends Exception { }

- 2) This My exception goes in the other section of the hierarchy.

* ~~Nested Try-Catch Exception~~

Exam Watch:

1)

```
try { throw new ArithmeticException(); }  
catch (Exception e) { Ex e = new AEC();  
    throw e; } // If and so  
// all blocks have already  
// been defined.  
this throws AE.
```

2) i) Class A has smart pointer

ii) { smart pointer because it has
 pvm(-)}

iii) smart pointer has & { }

iv) ArithmeticException e = nullified

try { } catch (Exception e) { }

throws; // Null pointer exception.

}

What makes you happy?

HappyCollegeDays

catch (exception e)

{

e1 = e;

}

}

In the above example if reference e at line 4 is a local variable and should be initialized to null.

As it is declared in the main func, the scope remains throughout the main and hence it should not be reused anywhere.

If e is written at line 2 outside the main func, it becomes instance variable which by default is initialized so we do not have to initialize it.

If a variable is declared inside a try block, its scope remains inside a try block and can be reused outside try block.

Same rule applies to catch and finally block as well.

- Ex. If a exception is thrown using a code, we can write statement after it, however if an exception is thrown manually, anything written after that is not reachable and generates compilation failure.

Ex. class A {

psvm(-) {

try {

throw new AE();

sop(" - "); X

}

int n=10;

catch (Ex e) { sop("catch"); }

EW

4) Method Propagation.

class A

public void m1() throws Exception {

m2();

}

private void m2() throws Exception {

m3();

3

private void m3() throws Exception {

throws new IOException();

3

3 A real 3

3

3 (→ wiring)

3 problem

X 3 (→ error 3)

3 (→ error 3)

class B extends A

{

psvm(-)

try{

new B().m1();

}

except

Catch(ex:e)

{

Sop("exception");

}

Output

exception 1

- Assertion's

Assertion's are used to ensure that the code does not execute ~~unintended~~ or process unwanted conditions.

Syntax:

`assert exp1: exp2; true`

Condition

Where

- i) `exp1` should return a boolean value.
- ii) `exp2` can be a string, arithmetic operation or just a variable.
- iii) `exp2` is optional and so can be skipped.
- iv) If we use mtd's as `exp1`, then it must return a boolean value.

If we use mtd as `exp2`, then it must return something preferably string, and cannot be void.

Note:

Assertion by default are disable by JVM, to enable them we write

java -ea <classname>

↓ (from file described in)

+ "enable" assertion step by step

at runtime.

(you can handle assertion error)

Base Rule:

- i) If exp1 evaluates to true, we ignore the complete statement.
- ii) If exp1 evaluates to false, we throw a red color assertion error, with display of exp2. [if available]
- iii) When the assertion is evaluate to false, program does not proceed further.

```
package ocpjp;
```

```
class A {
```

```
    int m = 1;
```

```
    void m1 (int marks)
```

```
    {
```

```
        assert m1() > m2();
```

```
        System.out.println("Marks are " +
```

```
marks);
```

```
}
```

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

```
{
```

```
    new A().m1(120);
```

```
    System.out.println(m1());
```

```
}
```

```
    boolean m1 ()
```

```
    {
```

```
        return false;
```

```
}
```

```
    of
```

```
showing A m2 ()
```

```
{
```

```
    return new A();
```

```
}
```

```
}
```

What makes you happy?
#HappyCollegeDays

O/P: Assertion error.

#

-) assert false;

+) assert true; ~~and for~~

+) assert $x == 10$; ~~and for~~

+) assert str == str; ~~and for~~

+) assert x <= 10; ~~and for~~

+) assert x >= 10; ~~and for~~

+) assert x < 10; ~~and for~~

+) assert x > 10; ~~and for~~

X . 0 < x + t \rightarrow 0 < 20 \rightarrow post

X . i + t < 20 \rightarrow 0 < 20 \rightarrow true

2003-07-02 10:00:00

first condition should was str !=

should & after put a good assert

+) assert str != "abc"; \rightarrow good

+) assert str == "abc"; \rightarrow good

+) assert str == "abc"; \rightarrow good

+) assert str != "abc"; \rightarrow good

+) assert str == "abc"; \rightarrow good

+) assert str != "abc"; \rightarrow good

+) assert str == "abc"; \rightarrow good

+) assert str != "abc"; \rightarrow good

Guidelines for Using Assertion:

- 1) Assertions should not be used to check the argument of method which is public, instead use illegal argument exception.
- 2) In assert statement, do not change the value of a variable.

Eg: assert ++x > 0; X
assert x > 0 : x++; X

Handling Assertion Error

- 1) We can handle assertion error using using a try catch block using fall
 - 1) Assertion error.
 - 2) Error.
 - 3) If throwable:

Note: fall.. assertion syntax are legal and can be used.

Inner Classes

Date

--	--	--

There are four types of inner classes.

- i) A class inside a class.

E.g.,

```
{ class A {  
    int x; void display();  
};  
void display() { cout << "A"; }  
class B {  
    int x; void display();  
};  
void display() { cout << "B"; }  
int main() {  
    A a = new A();  
    a->display();  
    B b = a->new B();  
    b->display();  
}
```

A a = new A();

A·B b = a·new B(); // A·B b = new A()·new B();

3

O/p: B

(iii) Static Inner Class.

Point

psvm (-)

a. A·B b = new A·B();

b. display ();

O/p : B.

(iii) Method Local Inner Class.
(A class created inside the method).

Ex → PTO

Class A

declaration of non final variable

error: final can't be used here

so int x; is a valid declaration

void display()

{

class B

{

~~int x;~~ final int y = 5 //must be declared as final.

void display()

{

cout << B^n; /> cout << y;

}

}

B b = new B();

b. display();

}

psvm (-)

{

new A(). display();

}

}

O/P : B

Date

--	--	--

Note: To use a variable declared inside the mtd in the class, we have to declare it as final.

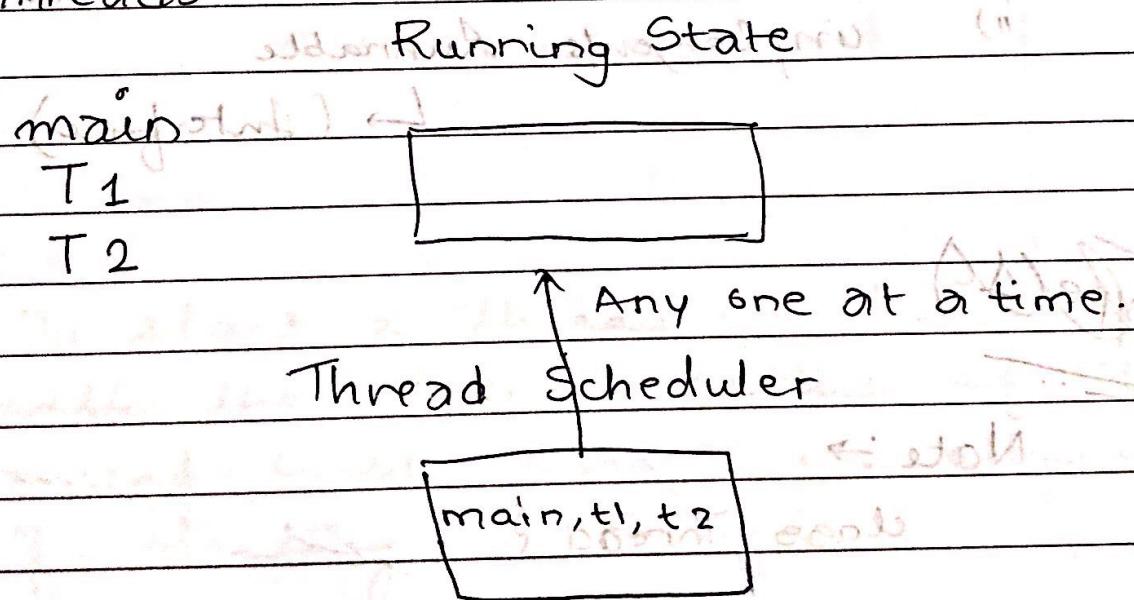
Multi-Threading :

A program by default always runs in main thread.

A programmer can start a new thread manually and perform its own execution independent of main thread.

Consider following scenario.

Threads



Syntax's for Creating and Starting

Thread

↳ Create a new thread

↳ Extend Thread class or extend

↳ Write its fields and constructor

→ If a class wants to participate
in multithreading, it has 2 choices.

i) extends Thread

↳ (Class)

ii) implements Runnable

↳ (Interface)

Note ⇒

class Thread { }

public void start()

run();

}

```
public void run()
{
```

heart attack A car is

}

must withstand 100 km per second

}

(from http://heartattackbond.com)

```
interface Runnable {
```

 void run();

↳ heart attack bond () took A car

steel 2

To start a thread we have to
call the `start()` method of the
thread class. There is no other way
of starting a thread.

at some speed NOT threads yet
so we can't get a lot of benefit at
some time all pillars got benefit of
with pizza but not benefit to
private as private

Type 1:

```
class A extends Thread  
{  
    public void run() // Overrides run.  
    {  
        System.out.println(Thread.currentThread().getName());  
    }  
}
```

```
public static void main(-)
```

```
{  
    new A().start(); // Starts New thread  
    Starts
```

in main method we'll start a thread
with new A().start(); in main thread.
parent to our thread sends command
} to start & pictures of
}

By default JVM assigns name to
the thread. We can assign our name
to thread by calling the constructor
of thread class and passing the
name as string.

~~Type 2~~

Using an object of thread class.

(more prot.) A

class A extends Thread { }

public void run() { }

System.out.println(Thread.currentThread().getName());

}

psvm (-) throws InterruptedException { }

}

Thread t1 = new Thread(new A(), "A");

Thread t2 = new Thread(new A(), "B");

t1.start();

t2.start();

main

5/0

A

9/0

t1.join(); // main will get executed after t1

t2.join(); // main will get executed after t2

Soot("Thank");

}

}

A (String name)

{ Local variable A starts

super (name);

psvm (-) equivalent command (C/C++)

{

new A ("A"). start();

new A ("B"). run();

}

}

O/P A or OR main

It will always print main even if it's a A

It will always print B because B is the object of list

(String -) good

Type 3:

Implements Runnable

class A implements Runnable

```
public class A implements Runnable {
    public void run() {
        System.out.println("A");
    }
}
```

So at (Thread.currentThread().getName());

(Current thread name);

}

(Parent thread's name) to

~~public stat~~

psvm()

{

Thread t1 = new Thread (new A(), "A")

("A") here it will use S1 baseT

t1.start();

}

}

(This) T here is baseT) to

(new A()

{

(2) last 3 if

'2 last 2 is #

Type 4:

Anonymous Way.

class A {

 public void m() throws InterruptedException
 {

 Thread t1 = new Thread("A")

 public void run()

 {

 System.out.println(Thread.currentThread().getName());

 }

}

 Thread t2 = new Thread("B")

{

 public void run()

{

 System.out.println(Thread.currentThread().getName());

}

};

 t1.start();

 t2.start();

}

What makes you happy?

#HappyCollegeDays

~~Type 5.~~

Anonymous using Runnable.

```
class A
{
    public void run() throws InterruptedException
    {
        System.out.println(Thread.currentThread().getName());
    }
}

Thread t1 = new Thread(r);
t1.start();
```

Type 6 :

```
class A {
```

 pvm() throws InterruptedException

```
{
```

```
    new Thread(new Runnable ()
```

```
        { public void run ()
```

```
            { ... }
```

```
}
```

```
        ). start ();
```

```
}
```

```
}
```

Thread Synchronization

There are 2 ways in which

(a) (Sync.) lock can be implemented

(b) (A var) --- = ST based

- 1) → synchronized Block [needs an object]
 (1) lock + t
- 2) → synchronized method. ST

Synchronized Block:

class A extends Thread

```
{  
    String str = " ";
```

```
    public void run ()  
    {
```

 synchronized (str) // takes an object.

```
{
```

```
    cout (Thread::currentThread ()-> getName ());
```

```
    cout ( _____ );
```

3

3

psvm() throws InterruptedException

Thread t1 = new Thread(new A(), "A");

Thread t2 = new Thread(new A(), "B");

t1.start();

t2.start();

Thread A starts

i = 0; print2

() new bio writing

exit || () background

() current thread

() two

{ }

Synchronized b Method

format

Threads display their own names sequentially

class A extends Thread

{ A() { } } extends Thread

{ "A" }

public void run()

{ A. m1(); }

{ "A" }

static synchronized void m1()

{ A. t1 = new Thread() { }

sout (Thread.currentThread(). getName());

try {

sleep (1000);

}

catch (Int ex)

{ }

Sout (Thread.currentThread(). getName());

}

public static void main () throws
InterruptedException

Thread t1 = new Thread (new A(),
"uA");

Thread t2 = new Thread (new A(),
"uB");

t1.start();

t2.start();

}

{loop goes}

}

{x2 int} loops

{ }

loop does not have any data