**OOPS WITH JAVA**

**Read all options again before ticking the mcq**

**num << n = num \* 2n**

**num >> n = num / 2n**

**~ 10 => -(n+1) => -11**

**~ -5 => -(n+1) => 4**

Interview important

* JDK & JRE
* Java Annotations
* Node Stream
* PrintWriter v/s BufferedWriter && Scanner v/s BufferedReader
* Object Stream

When is downcasting required ?

Iff you are invoking sub class specific functionality (eg : study or teach), using super class reference .

eg : Person p=new Student(....);//up casting : done impl by javac : no err

sop(p);//p.toString() : no javac err : since Person class has toString method.

p.study();//javac err ! , javac checks it by : Type of the ref (it chks for study in Person class)

Soln : to satisfy javac -- down casting(equivalent to : narrowing)

((Student)p).study();//no javac n no run time err

p=new Faculty(...);//no javac err : up casting !

p.teach();//javac err

((Faculty)p).teach(); //works !

((Student)p).study();//run time err : java.lang.ClassCastException :Faculty CAN NOT be cast into a Student!

Solution : instanceof : keyword in java

Is पोरगा instance of बाप – ret true

What will be o/p ?

Emp e =new Mgr(...);//up casting

e instanceof Mgr -true

e instanceof Emp --true

e instanceof Object --true

e instanceof SalesMgr -- false

e instanceof Worker -- false

e=null;

Important statement

**Java compiler resolves method binding by type of the reference & JVM resolves it by the type of the obejct, reference is referring to.**

**Special note on protected**

**Protected members act as default scope within the same package.**

**BUT outside pkg -- a sub-class can access it through inheritance(i.e just inherits it directly) & CAN'T be accessed by creating super class instance.**

Annotations are compile time , internally written as interface, removed from .class file

Downcast needed whne you are accesing subclass function using superclass

In Java , abstraction is already achieved using unit of encapsulation : class

When you define methods(functionality) in the class , it's user(Client code using these methods) does not need to know about the actual implementation of the methods, it just needs to know about the invocation.

eg : public abstract double area();//legal syntax

private abstract double area();//javac error

Rules :

Any time a class has one or multilple abstract methods ---- class must be declared as abstract class.

eg. public abstract class BoundedShape {....}

Abstract classes can't be instantiated BUT can create the reference of abstract class type to refer to concrete sub-class instances.

eg : BoundedShape shape=new Rectangle(....);//legal

BoundedShape shape2=new BoundedShape();//javac err : RHS

Abstract classes CAN HAVE concrete(non-abstract) methods.

Abstract classes MUST provide constructor/s to init its own private data members.(for creating concrete sub class instance)

eg : Emp : empId, dept...: private : abstract

Mgr extends Emp : to init empId, dept ... : MUST supply a constr in Emp class.

Can a class be declared as abstract & final ? NO

final -- keyword in java

Usages

1 final data member(primitive types) - constant.

eg -- public final int data=123;

2. final methods ---can't be overridden.

usage eg public final void show{.....}

This show() method CAN NOT be overridden by any of the sub classes

eg -- Object class -- wait , notify ,notifyAll

3. final class --- can't be sub-classed(or extended) -- i.e stopping inheritance hierarchy.

eg -- String ,StringBuffer,StringBuilder

eg : public class MyString extends String {...} //javac err

4. final reference -- references can't be re-assigned.

eg --final Emp e=new Mgr(.......);//up casting

e=new Worker(.....);//compiler err

Interface in Java

Java Interface also represents IS-A relationship , with the implementation class

The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

Syntax:

default(no modifier)/public interface NameOfInterface extends comma separated list of super interfaces

{

//data members --- public static final : added implicitly by javac

int DATA=100;

//methods -- public abstract : added implicitly by javac

double calc(double d1,double d2);

}

Implementing class syntax

default(no modifier)/public class NameOfClass extends SuperCls implements comma separated list of interfaces {

//Mandatory for implementation class to be non-abstract(concrete): MUST define/implement all abstract methods inherited from all i/fs.

}

eg : public class Circle extends Shape implements Computable,Runnable {...}

Q . What is a functional i/f

An interface containing single abstract methods (SAM)

Q) What is marker or tagged interface?

An interface that has no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM(Run time marker) so that JVM may perform some useful operation.

Abstract class doesn't support multiple inheritance. Interface supports multiple inheritance.

In order for a class to implement an interface, it must implement all its declared methods. However, a class may not implement all declared methods of an abstract class. Though, in this case, the sub-class must also be declared as abstract.

Abstract classes can implement interfaces without even providing the implementation of interface methods.

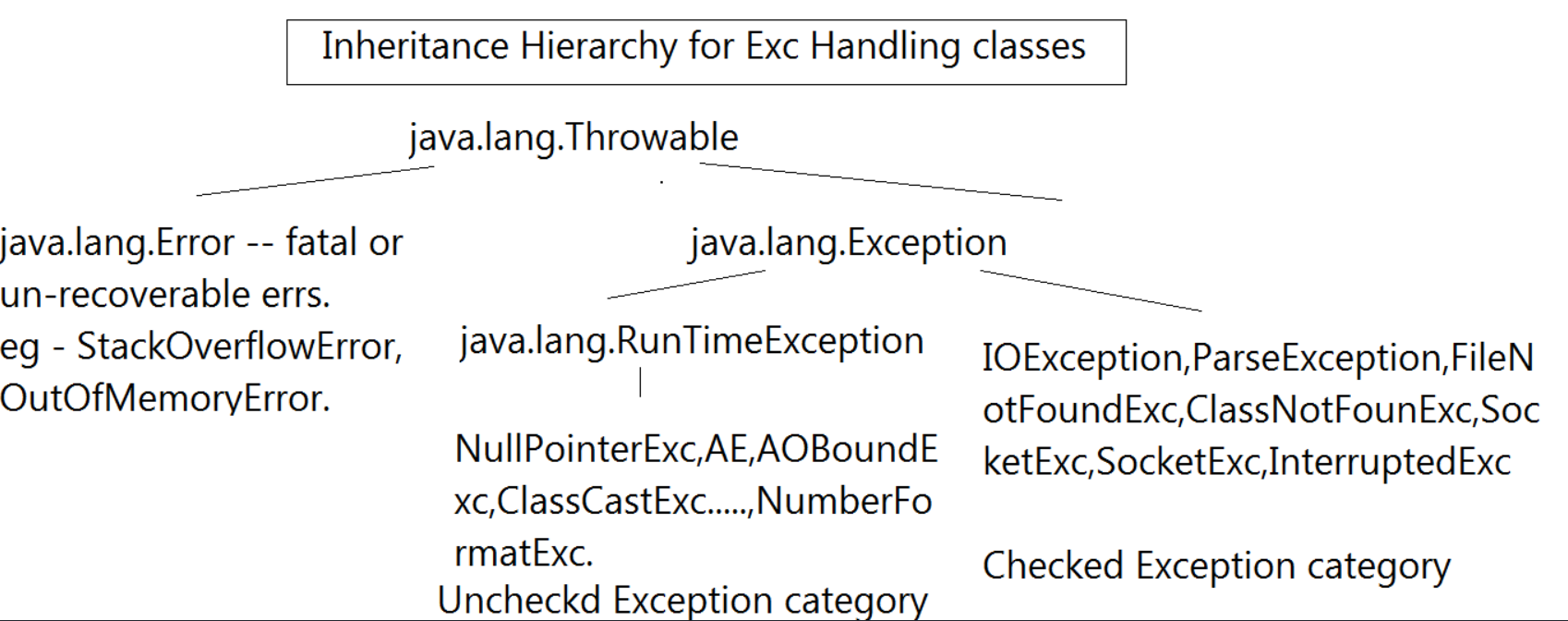
An interface is absolutely abstract and cannot be instantiated, doesn't support a constructor. An abstract class also cannot be instantiated BUT can contain a constructor to be used while creating concrete(non abstract) sub class instance.

throws --- keyword meant for javac

Meaning -- Method MAY raise specified exc.

Current method is NOT handling it , BUT its caller should handle.

Mandatory--- only in case of un handled(no try-catch) chked excs(not extended from RuntimeException).



finally -- block -- finally block ALWAYS survives(except System.exit(0) i.e terminating JVM)

if try aborted and no matching catch block is present then code is aborted but finally if present will be executed

5.3 try {...} finally {....}

try-with-resources block

From Java SE 7 onwards --- Java has introduced java.lang.AutoCloseable -- i/f

It represents --- resources that must be closed -- when no longer required.

Autocloesable i/f method

Creating Custom Exception(User defined exception or application exc)

1. Create a pkged public class which extends Throwable(not reco but legal)/Exception(recommended)/Error(not reco but legal)/RuntimeExc(not reco but legal)

eg : public class MyException extends Exception{

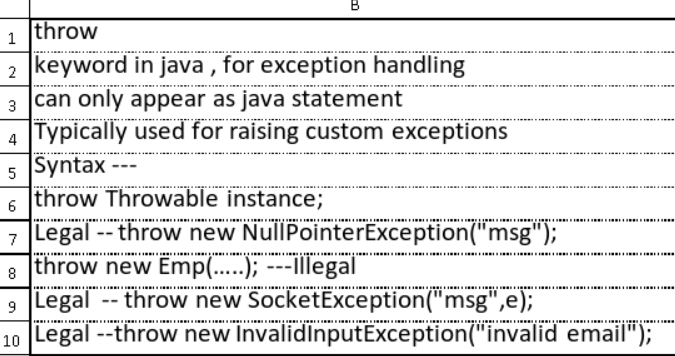
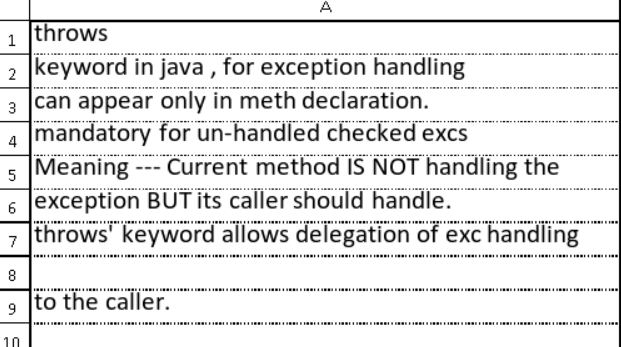
public MyException(String mesg)

{

super(mesg);

}

}



Strings

Important String class constructors

1.String(byte[] bytes) --- byte[] ----> String converter

2.String(char[] chars) --- char[] ---> String converter

3.String (byte[] bytes,int offset,int len) ---byte[] ----> String converter from the specified offset , specified len no of bytes will be converted.

eg . String s=new String(bytes,3,4); --- String will contain bytes[3]----bytes[6]

The String Literal Pool

it's a collection of references to String objects. Strings, even though they are immutable, are still objects like any other in Java. Objects are created on the heap and Strings are no exception. So, Strings that are part of the "String Literal Pool" still live on the heap, but they have references to them from the String Literal Pool.

Yeah, so that doesn't really explain what the pool is, or what it's for, does it? Well, because String objects are immutable, it's safe for multiple references to "share" the same String object. Take a look at this example:

Var args

Legal ---

void doStuff(int... x) {

//B.L

} Illegal: javac error

void doStuff4(int x...) {....} // bad syntax : javac err

void doStuff5(int... x, char... y) {...} // too many var-args

void doStuff6(String... s, byte b) { } // var-arg must be last

syso(“%d”, 23) //err as no object, if string was this then as string is object then works

please read don’t forget to do regular expressions

Enum

public/default enum Direction {

EAST,

WEST,

NORTH,

SOUTH //optionally can end with ";"

}

Here EAST, WEST, NORTH and SOUTH are implicitely of type

public final static Direction EAST=new Direction("EAST",0) ---super("EAST",0);

public final static Direction WEST=new Direction("WEST",1) ---super("WEST",1);

Super class of all enums

public abstract class Enum<E extends Enum<E>>

extends Object

implements Comparable<E>, Serializable

When declared inside a class, enums are always static by default

eg public class TestOuter

{

enum Direction

{

EAST,

WEST,

NORTH,

SOUTH

}

}

To access a direction -- use TestOuter.Direction.NORTH.

constructor , Enum(String name,int ordinal)

enum Direction {

// Enum types

EAST(0), WEST(180), NORTH(90), SOUTH(270);

// Constructor

private Direction(final int angle) {

this.angle = angle;

}

// Internal state

private int angle;

public int getAngle() {

return angle;

}

}

BUT u can't instantiate enums using these constructors , since they are implicitely private.

You can override toString BUT you can't override equals since it's declared as final method in enum.

Generics

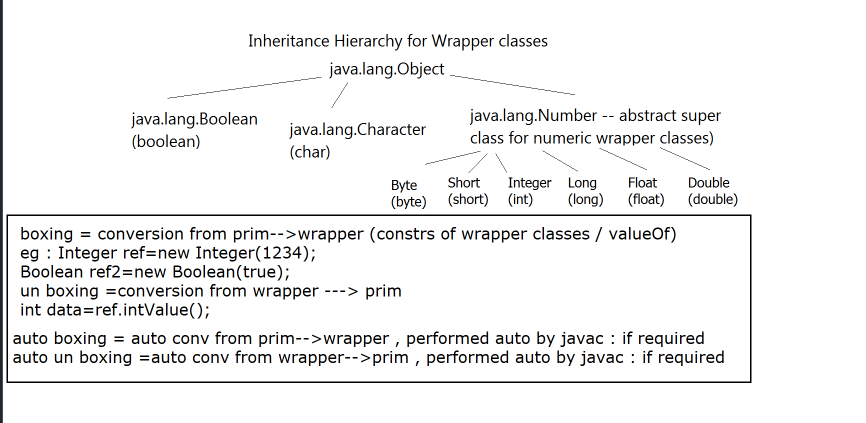
Advantages

Adds Type Safety to the code @ compile time

Meaning :

1. Can add type safe code where type-mismatch errors(i.e ClassCastExceptions) are detected at compile time.

2. No need of explicit type casting, as all casts are automatic and implicit.



Illegal: javac error

void doStuff4(int x...) {....} // bad syntax : javac err

void doStuff5(int... x, char... y) {...} // too many var-args

void doStuff6(String... s, byte b) { } // var-arg must be last

| String | StringBuilder | StringBuffer |

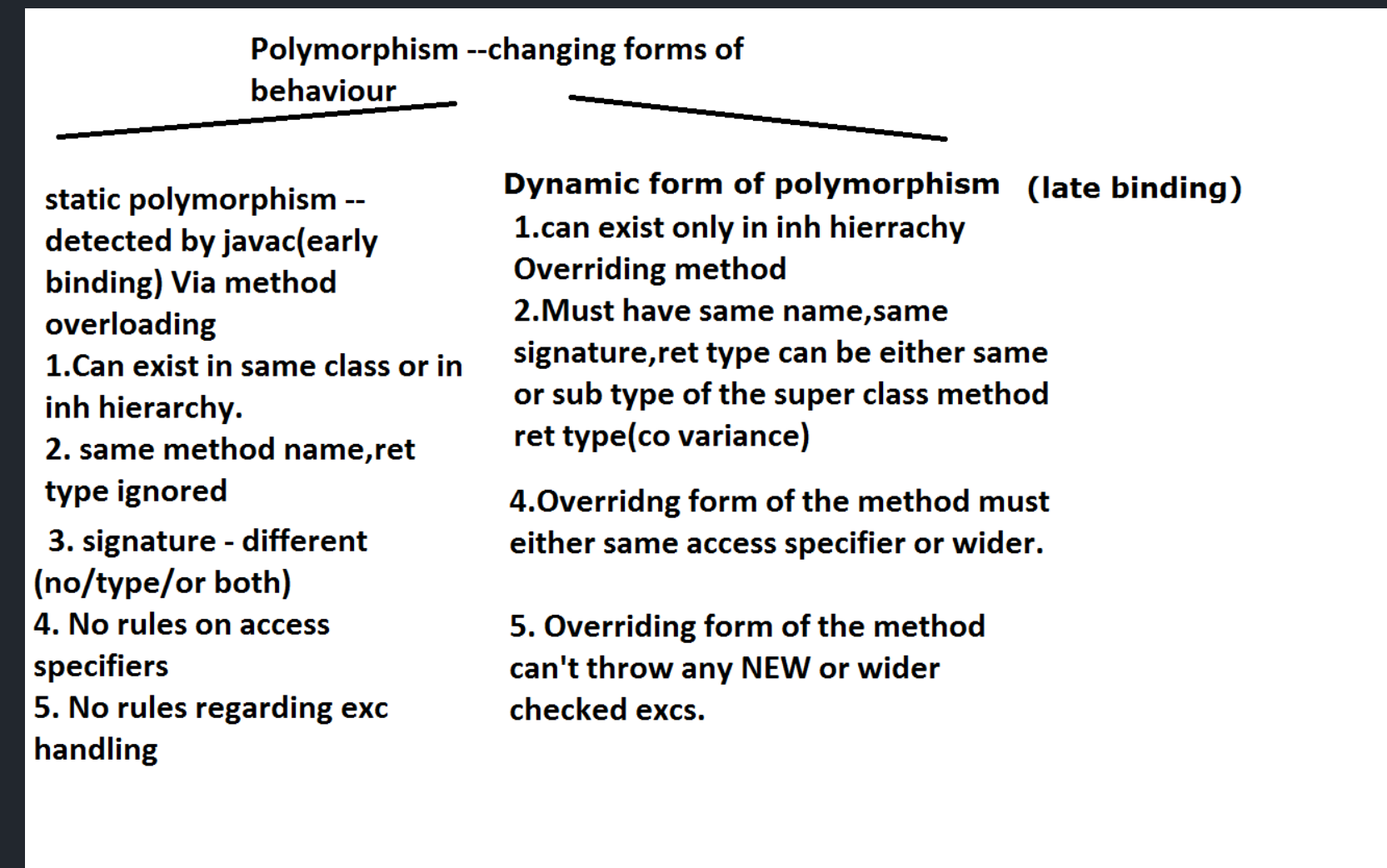
|  |  |  |
| --- | --- | --- |
| String | StringBuilder | StringBuffer |
| Immutable char seqnc | Mutable char seqnc | Mutable char seq |
| Inherently thread safe | Inherently thread unsafe | Inherently thread safe |
|  | More efficient than stringbuffer |  |
| Supports length() | Supprts length(), caa |  |

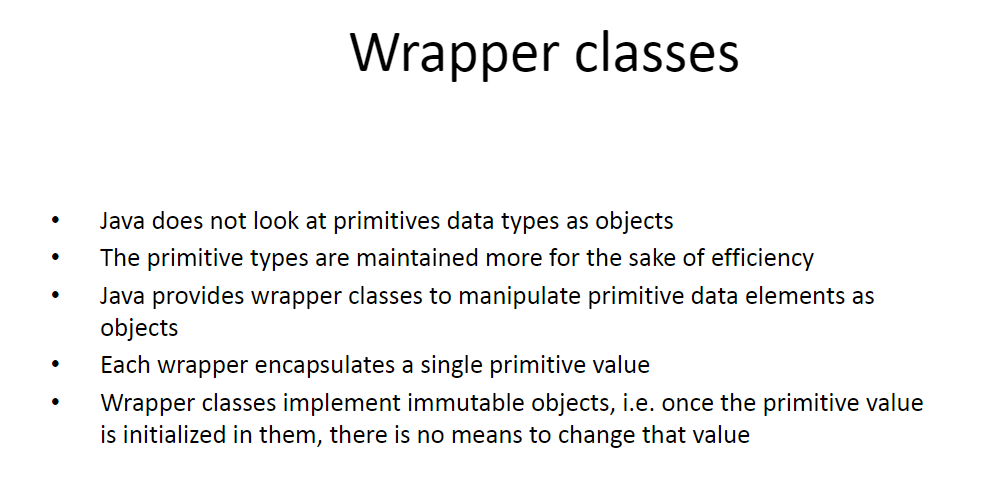
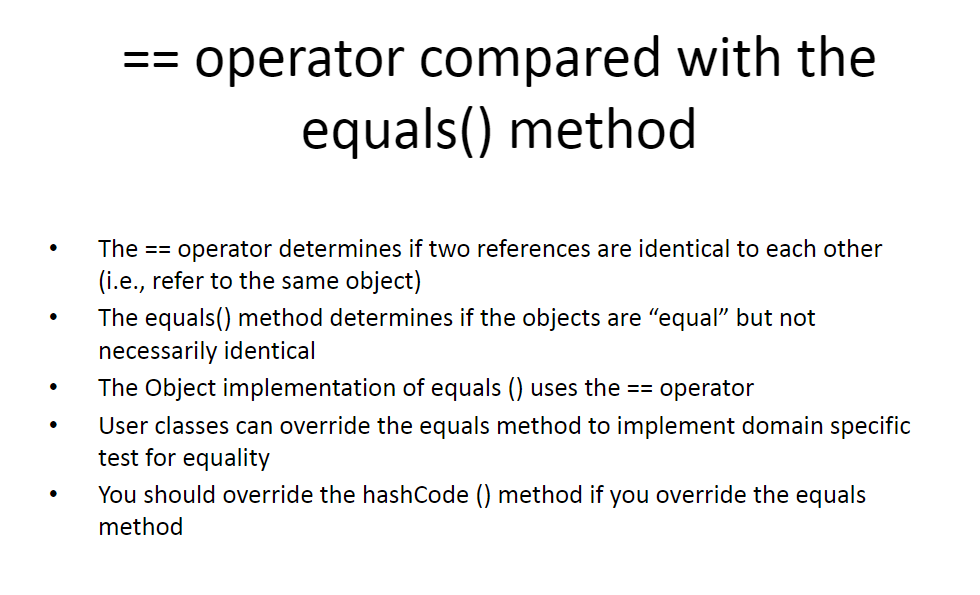
Date/Time Handling in Java(legacy API)

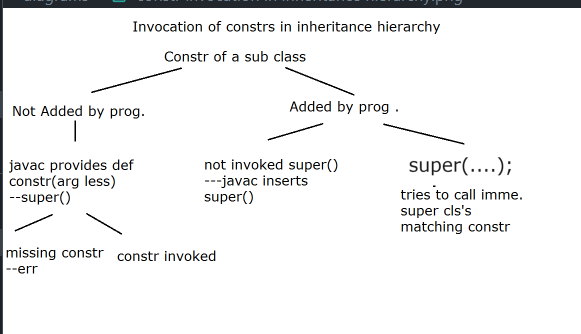
eg : SimpleDateFormat sdf=new SimpleDateFormat("dd-MM-yyyy , hh:mm:ss");

OR

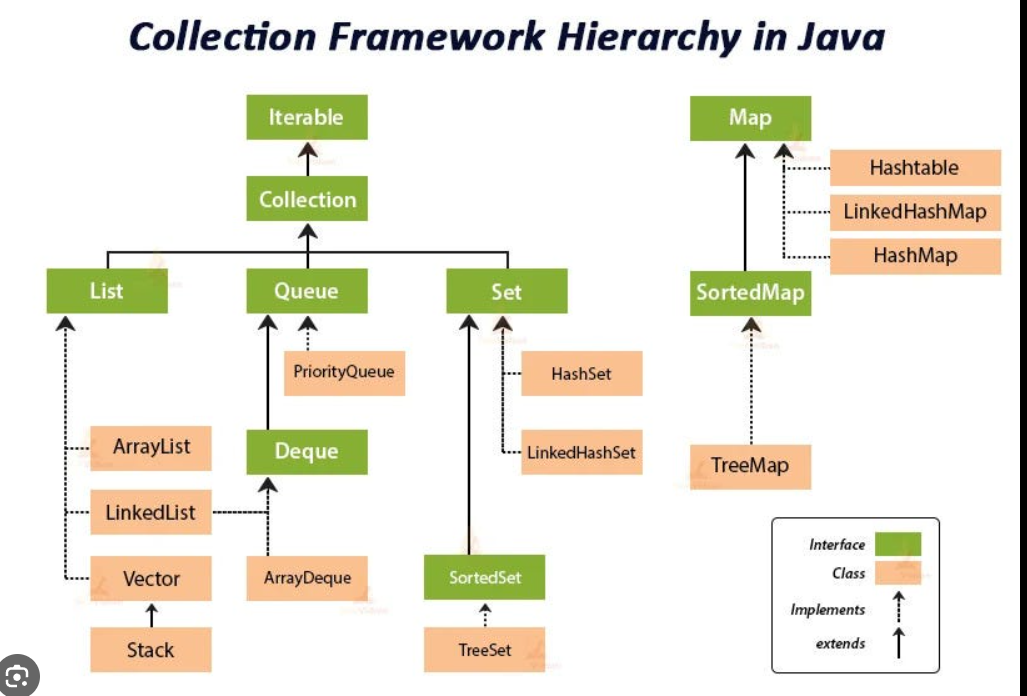
SimpleDateFormat sdf=new SimpleDateFormat("dd/MM/yyyy");







# Collection Framework





List<E> features

1. List represents ordered collection --- order is significant(It remembers the order of insertion)

2. Allows null references

3. Allows duplicates

4. Supports index based operation

ArrayList() -- default constructor. -- creates EMPTY array list object , with init capacity=10,size=0;

public ArrayList(int capacity) -- -- creates EMPTY array list object , with init capacity=capacity,size=0;

2. add methods

boolean add(E e) --- append

void add(int index,E e) --- insert

void addAll(Collection<E> e) -- bulk append operation

eg : l1 --- AL<Emp>

l1.addAll(.....);

AL,LL,Vector --- legal

HS,TS,LHS --legal

HM,LHM,TM --illegal --javac error

2.5 Retrieve elem from list

E get(int index)

Regarding exceptions with Iterator/List

1. java.util.NoSuchElementException -- thrown whenever trying to access the elem beyond the size of list via Iterator/ListIterator 's next() method.

2. java.lang.IllegalStateException --- thrown whenever trying to remove elem before calling next().

3. java.util.ConcurrentModificationException-- thrown typically --- when trying to use same iterator/list iterator --after structrually modifying list(eg add/remove methods of list)

Don’t use list methods to update list use iterators method then we don’t get error

java.lang.IndexOutOfBoundsException -- thrown typically -- while trying to access elem beyond size(0---size-1) --via get/set/remove methods of List.

4. search for a particular element in list

boolean contains(Object o)

NOTE : Based upon public boolean equals(Object o)

5. searching for 1st occurrence

use -- indexOf

int indexOf(Object o)

rets index of 1st occurrence of specified elem. Rets -1 if elem not found.

searching for last occurrence

use -- lastIndexOf

int lastIndexOf(Object o)

rets index of last occurrence of specified elem. Rets -1 if elem not found.

5.5

E set(int index,E e)

Replaces old elem at spepcified index by new elem.

Returns old elem

6. remove methods

E remove(int index) ---removes elem at specified index & returns removed elem.

boolean remove(Object o) --- removes element specified by argument , rets true -- if elem is removed or false if elem cant be removed.

**Natural Sorting**

Steps for Natural ordering

Natural Ordering is specified in generic i/f

**java.lang.Comparable<T>**

T -- UDT , class type of the object to be compared.

eg -- Emp,Account , Customer

I/f method

**int compareTo(T o)**

Steps

1. UDT must implement Comparable<T>

eg : public class Account implements Comparable<Account>

2. Must override method

public int **compareTo(T o)**

{

use sorting criteria to ret

< 0 if this < o,

=0 if this = o

> 0 if this > o

}

3.Use java.util.Collections class API

Method

public static void sort(List<T> l1)

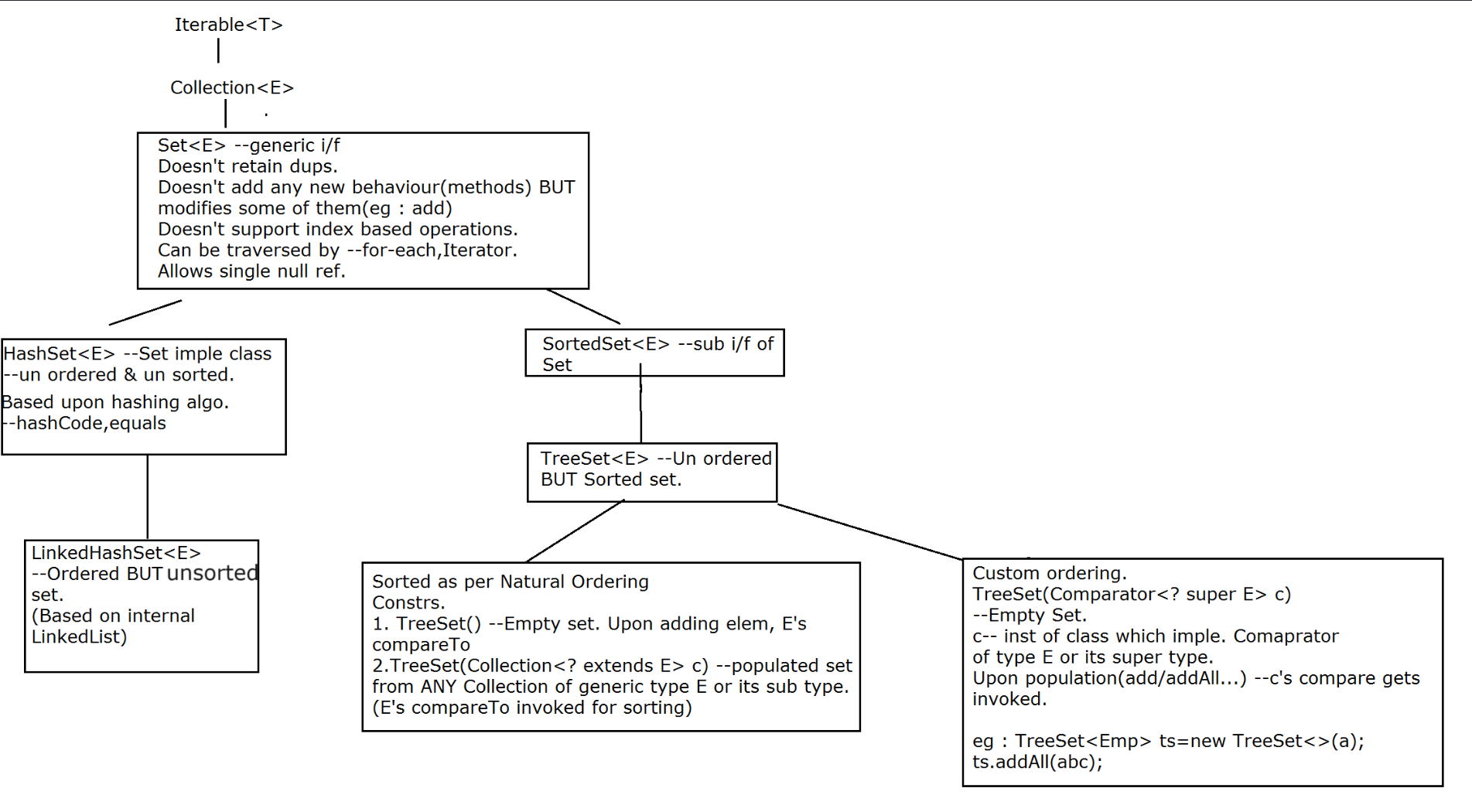
l1 -- List of type T.

**sort method internally invokes compareTo method(prog supplied)** of UDT & using advanced sorting algorithm , sort the list elems.

Limitation of natural Ordering

Can supply only 1 criteria at given time & that too is embedded within UDT class definition

Custom Ordering



What is Hashing?

Hashing is defined as the process of transforming one value into another based on a particular key.

OR

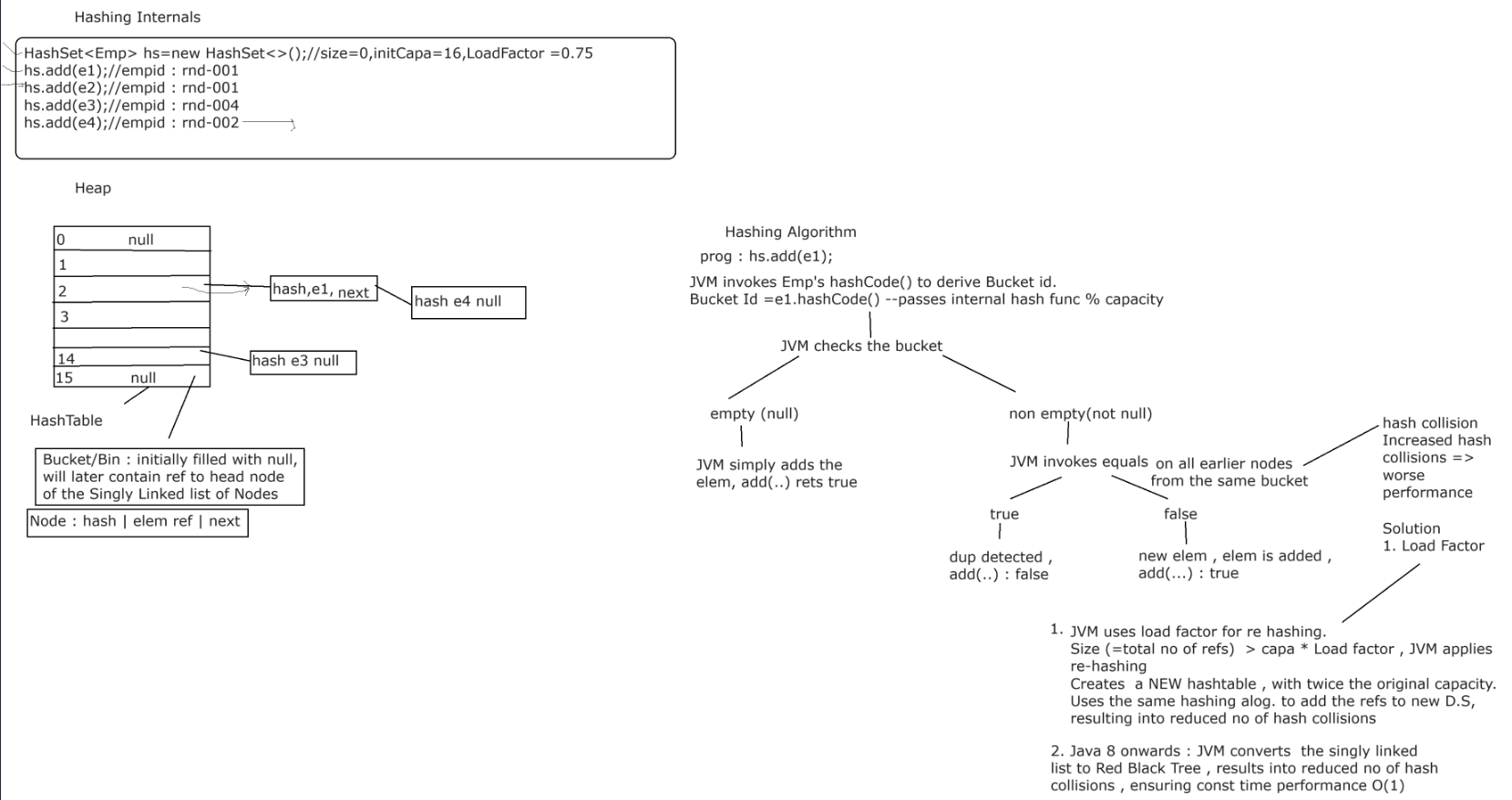
It can be also defined as transforming arbitrary sized data to a fixed size , typically an int value.

Why there is a guarantee that a duplicate ref can't exist in yet another bucket ?

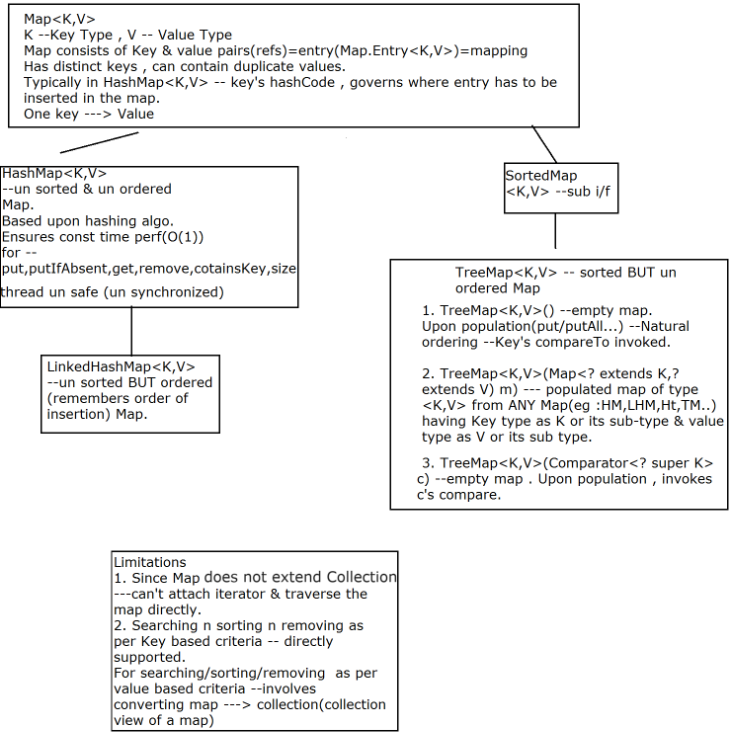
Answer is thanks to the contract between overriding of hashCode & equals methods

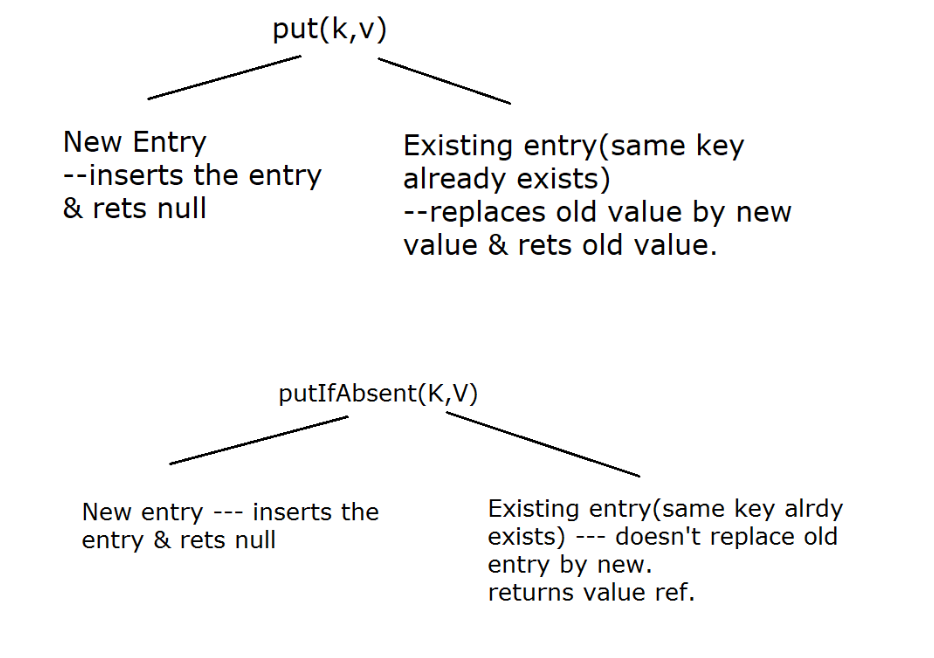
If two elements are the same (via equals() returns true when you compare them), their hashCode() method must return the same number. If element type violate this, then elems that are equal might be stored in different buckets, and the hashset would not be able to find elements (because it's going to look in the same bucket).

If two elements are different(i.e equals method rets false) , then it doesn't matter if their hash codes are the same or not. They will be stored in the same bucket if their hash codes are the same, and in this case, the hashset will use equals() to tell them apart.



map





1. Generic Method

What is it ? A Method which has it's own type parameter.

Can it exist in a non generic class? : Yes

Generic method example : Arrays.toList , Collections.sort

java.util.Arrays class

public static <T> List<T> asList(T... a)

Generic method from a non generic class(Arrays)

Type declaration fits between method modifiers n ret type

i/p : T... a : This method can accept : no args or T[] or T t1,T t2....(var-args of type T)

o/p : FIXED size list

How will you use it for getting a fixed size List<Integer> ?

eg : List<Integer> list=Arrays.asList(10,20,30,40,50);

Another example : Collections class : non generic class

public static <T extends Comparable<? super T>> void sort​(List<T> list)

Generic syntax :

? : wild card in generic syntax (it can be replaced by ANY type) : un bounded wild card

extends : Represents upper bound

super : Represents lower bound

? extends E : ANY type E or its sub type

? super E : E or its super type

eg : ? extends Emp => Emp or it's sub type(Mgr,Worker.....)

? super Mgr => Mgr or it's super type (Emp , Object)

public static <T> void sort(List<T> list,Comparator<? super T> c)

generic method :

where does type declaration fit ? : It's placed between method modifiers n ret type

More Details

? : wild card (represents ANY unknown type) => un bounded wildcard

**extends => upper bound (type of the upper bound super class / interface)**

**If your collection(list/set/map) is acting as a producer (of data) i.e while using retrieve operation, use upper bound**

**eg : ? extends Number => Number or it's subtype (Byte,Short.....Double)**

**get : type of the data that you get Number or it's subtype, BUT you can't add Integer , if it's collection of Double**

**Can be accessed using Number type of the ref.**

**(Producer extends)**

**super : lower bound**

**? super T => T or it's super type**

**Use it whenever your collection is acting as a consumer (data sink) : i.e wheneve you want to add data , to a collection.**

**(Consumer super)**

Why generics was introduced in Java ?

To add type safety @ compile time , so that explicit downcasting won't be required n code won't result into ClassCastException.

Javac can catch type mismatch errors

It also provides flexibility

So java compiler performs : type erasure , during compilation.

What is it ?

It is a process in which compiler replaces a generic parameter with actual class .

In type erasure, compiler ensures that no extra classes are created and there is no runtime overhead.

(avoids code bloats)

It adds the backward compatibility with legacy code(non generic , raw types)

Simple Rule

During the type erasure process, the Java compiler erases all type parameters and replaces each with its first bound if the type parameter is bounded, or Object if the type parameter is unbounded.

If there are multiple bounds , then Compiler will replace it by the 1st bound

Eg public class Node<T extends Comparable & Serializable> 🡺 comparable used

Producer consumer

Now answer this :

public static double sumOfSalaries(List<? extends Emp> list) {

double sum=0;

for (Emp e : list)

sum += e.computeSalary();

list.add(...);//what can be the legal replacement ?

return sum;

}

1. Emp

2. Mgr

3. SalesMgr

4. Object

5. Worker

6. HRMgr

7. TempWorker

8. Date

9. LocalDate

10. PermanentWorker

Ans : none of above !

Why ? : Since the method arg : List<? extends Emp> => caller can pass ANY List of Emp or any of it's sub types (eg : arg : List<Mgr> : u can't add a worker, Emp , Object to this list

List<PermanentWorker> " u can't pass Emp , Mgr , Worker....

So what's the bottom line ?

When the argument : uses extends (upper bound) : Collection<? extends E>

The Collection acts as read only . You CANT add any elems to it !

o.w : javac err!

PECS : Producer extends Consumer supers

Producer(Reader) => Collection acting as a producer of data ,i.e you can only access(Read) data

extends

Consumer(Writer) => Collection acting as a consumer of data ,i.e you can only add elems

super

eg : Java API eg :

Collections.addAll

public static <T> boolean addAll​(Collection<? super T> c, T... elements)

=> Collection : consumer (super)

T : Orange

boolean addAll​(Collection<? super Orange> c, Orange... elements)

Legal method arg : Any Collection(any List / any Set) of generic type : Orange | Fruit | Object

To such a Collection : what all can u add ? Orange or it's subtypes (eg : Mandarine)

eg : MUST Collections.copy(Collection<? super T> dest ,Collection<? extends T> src)

Java 8 new features

Addition of "default" keyword to add default method implementation , in interfaces.

Java 8 onwards allows us to add non-abstract method implementations to interfaces by using the default keyword. This feature is also known as Extension Methods.

1.default

Implementation class can either inherit this default implementation OR if needed , can override the same.

In case of ambiguity or to refer to default imple. from i/f -- use InterfaceName.super.methodName(...) syntax

Javac makes it mandatory to override the method , in case of dup default methods.



2. Can add static methods in java interfaces

**Such static methods can't be overridden in implementation class.**

**BUT can be re-declared.**

They have to be invoked using interface name , even in implementation or non implementation classes.(otherwise compiler error)

eg :

interface Formula {

double calculate(double a);//public n abstract : added implicitly by javac and if variable then adds public static final

//public : added implicitly by javac

default double sqrt(double a,double b) {

return Math.sqrt(a+b);

}

//javac adds implicity : public

static boolean testMe(String mesg)

{

//some imple. logic

}

}

API eg : List.of methods

3. Functional programming In java

Functional interfaces (java.util.function) -@FunctionalInterface

An interface which has exactly single abstract method(SAM) is called functional interface.

eg Runnable,Comparable,Comparator,Iterable,Consumer,Predicate,Supplier,Function...

public interface A { double calc(int a,int b);} : YES (SAM: calc)

public interface B extends A {} : YES (SAM: calc)

public interface C extends A { void show();} : NO 2 abstract methods

public interface D {} : marker i/f

public interface E extends A {

default void show(){}

static void test() {...}

} : YES (SAM : calc)

4. Lambda

1st language to use lambda 🡺 LISP

Following are some examples of Lambda expressions.

1.(int a, int b) -> { return a + b; }

OR can be reduced to

(int a, int b) -> a + b

OR further can be reduced to

(a,b) -> a+b

2. () -> System.out.println("Hello World")

3. s -> System.out.println(s)

4. () -> 42

5. () -> 3.1415

Main Differences between Lambda Expression and Anonymous class

1. One key difference between using Anonymous class and Lambda expression is the use of "this" keyword.

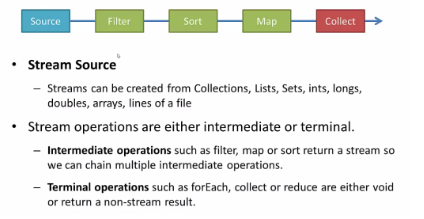
For anonymous class this keyword resolves to anonymous class, whereas for lambda expression this keyword resolves to enclosing class where lambda is written.

2. Another difference between lambda expression and anonymous class is in the way these two are compiled.

Java compiler compiles lambda expressions and convert them into private method of the class.

(written by mote ->) Java compiler compiles lambda expressions and convert them into private method of the class. and hence in lambda if we write a method or something which has checked exception then it gives error even if we have out big try catch block. to solve this we have to include another inner try catch to handle that exc.

Streams (java.util.stream)



Streams are wrappers (abstraction) around a data source(eg : array , collection ,lines from a text file....) .

A stream represents sequence of elements from a data source .

It supports : 2 different type of streams

Streams holding primitive types : IntStream , LongStream , DoubleStream (interfaces)

Streams holding referece types(T) : Stream<T> (interface)

Important Facts of Java 8 Streams:

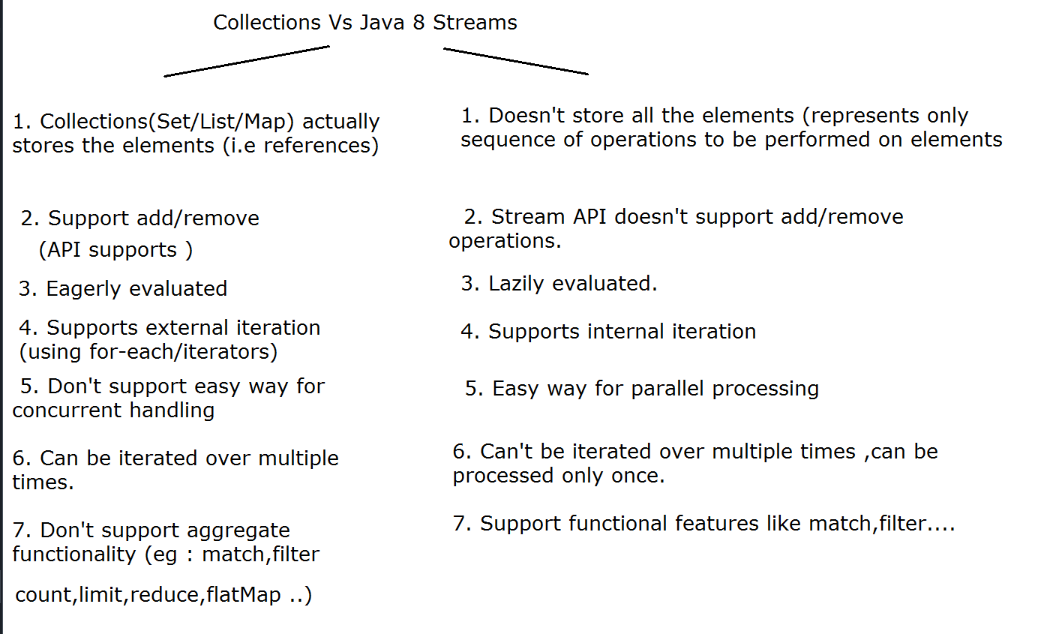
1. Streams are implicitely closed , after terminal operation (i.e they can't be re used after terminal operation)

Otherwise throws --IllegalStateException (reason :stream has already been operated upon or closed)

Where as , collections are re-usable.

2. Streams follow , vertical execution order.

3. Streams support lazy evaluation (meaning none of the intermediate operations are performed , until its closed by terminal operation)



**Do day15 ccee practice codes**

required stream classes --- BR(ISR(System.in))

Alternative is --- use Scanner class.

Adv. of Scanner over above chain ----- contains ready-made parsing methods(eg --- nextInt,nextDouble.....)

But Scanner is not Buffered Stream

Can combine both approaches.(new Scanner(br.readLine())

java.io.PrintWriter --- char oriented buffered o/p stream --- which can wrap any device.(Binary o/p stream or Char o/p stream)

Constructors---

PrintWriter(Writer w) --- no auto flushing,no conversion, only buffering

PrintWriter(Writer w, boolean flushOnNewLine)--- automatically flush buffer contents on to the writer stream --upon new line

PrintWriter(OutputStream w) --- can wrap binary o/p stream -- buffering +conversion(char-->binary),no auto-flush option

PrintWriter(OutputStream w , boolean flushOnNewLine) ---

API Methods----print/println/printf same as in PrintStream class(same type as System.out)

Stream class which represents --- Char o/p stream connected to Text file. --- java.io.FileWriter

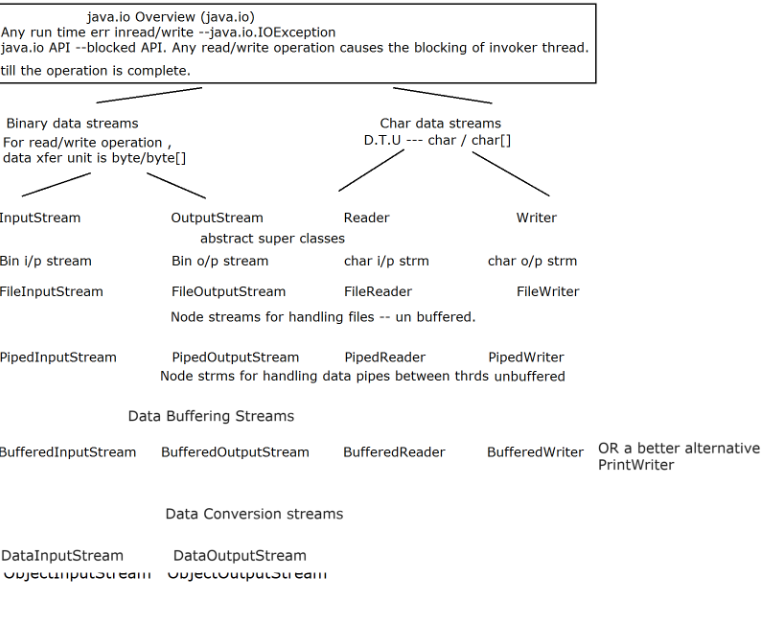
Constructor

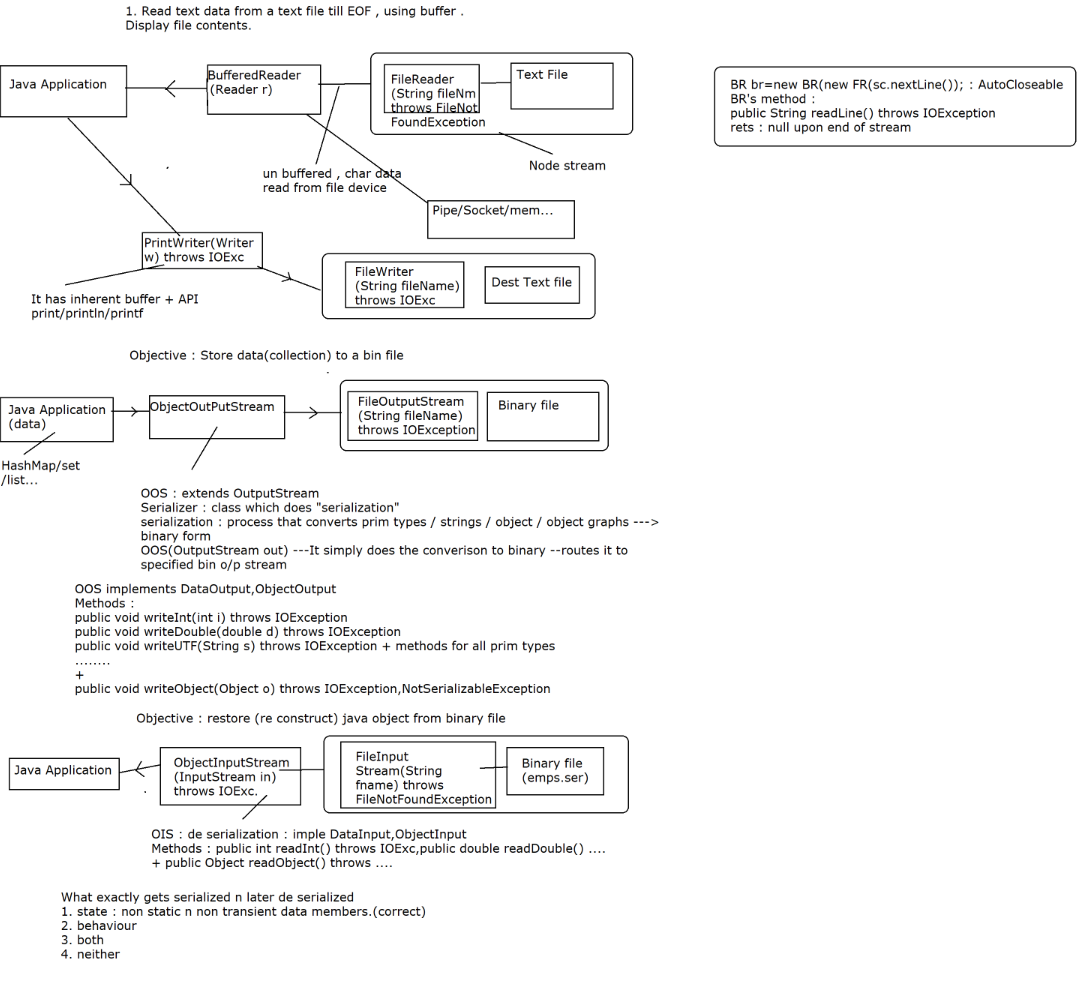
FileWriter(String fileName) throws IOException -- new file will be created & data will be written in char format.

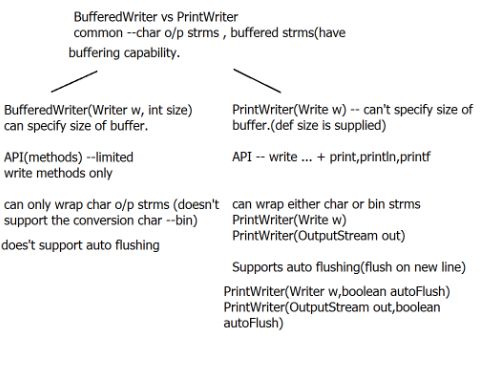
FileWriter(String fileName,boolean append) --- if append is true , data will be appended to existing text file.

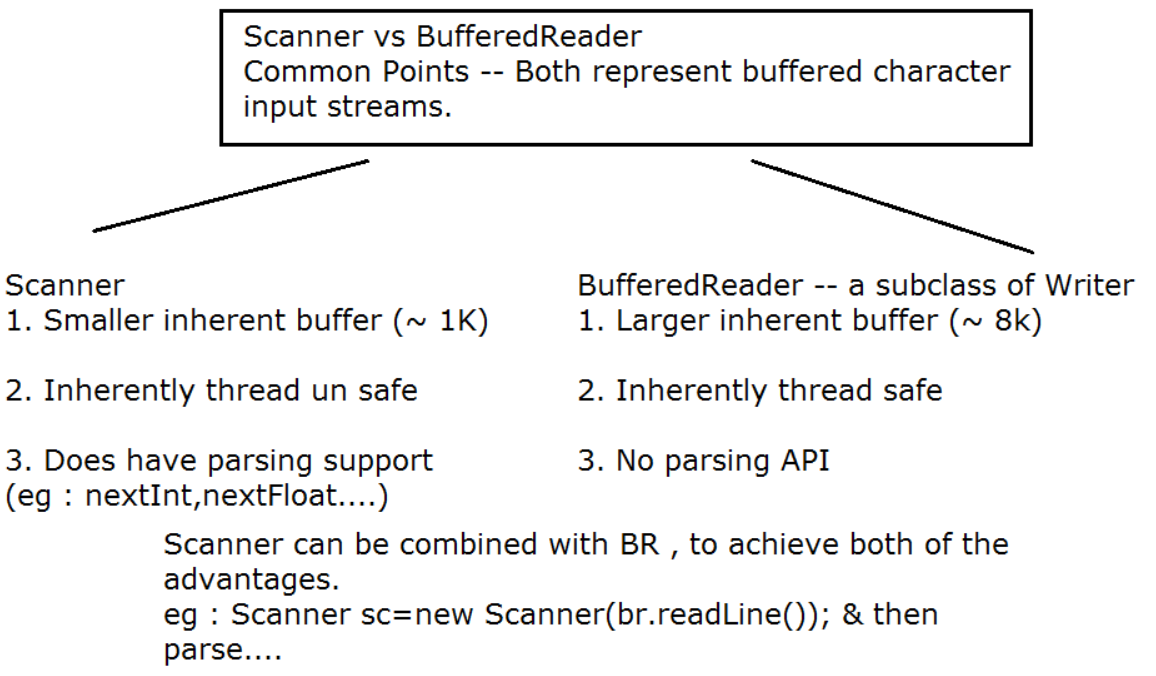
serialization= extracting state of object & converting it in binary form.

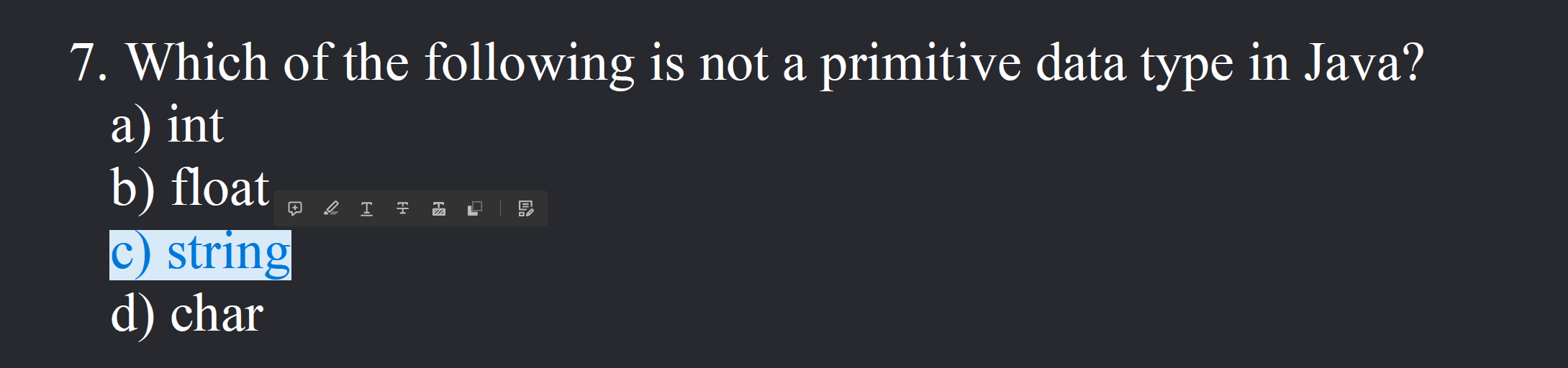
state of object = non-static & non-transient data members

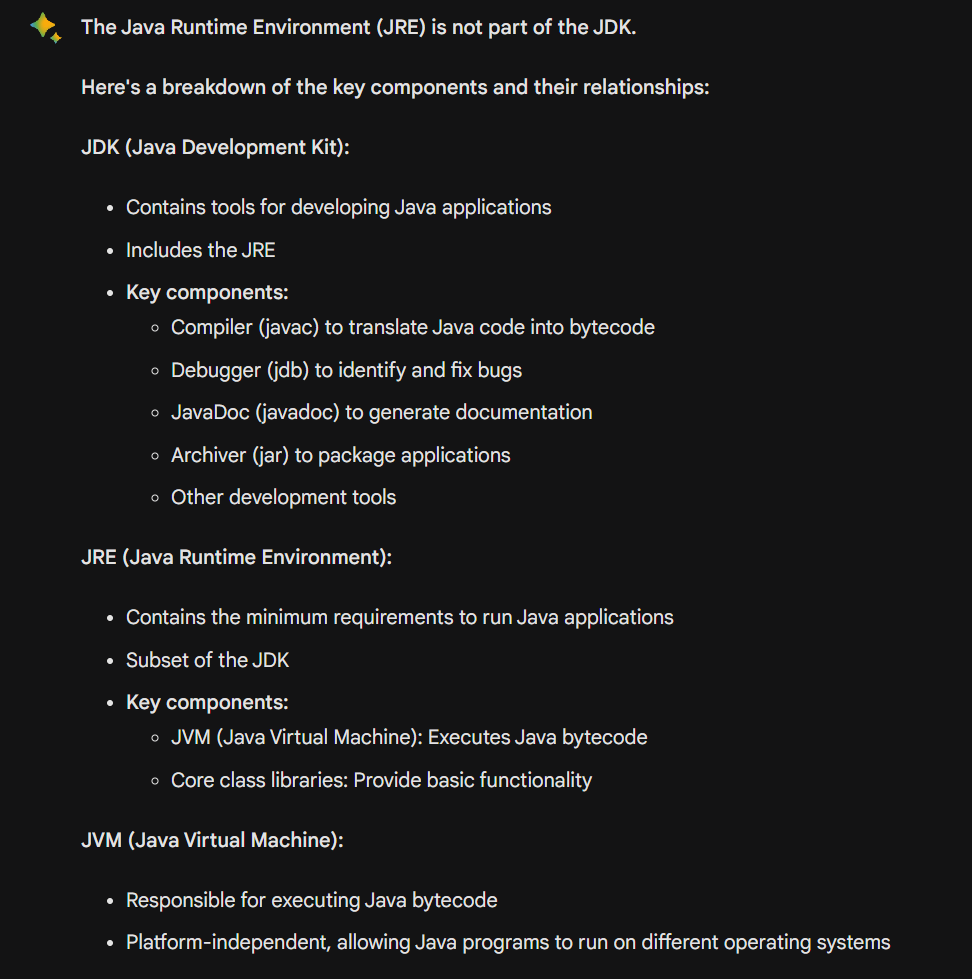


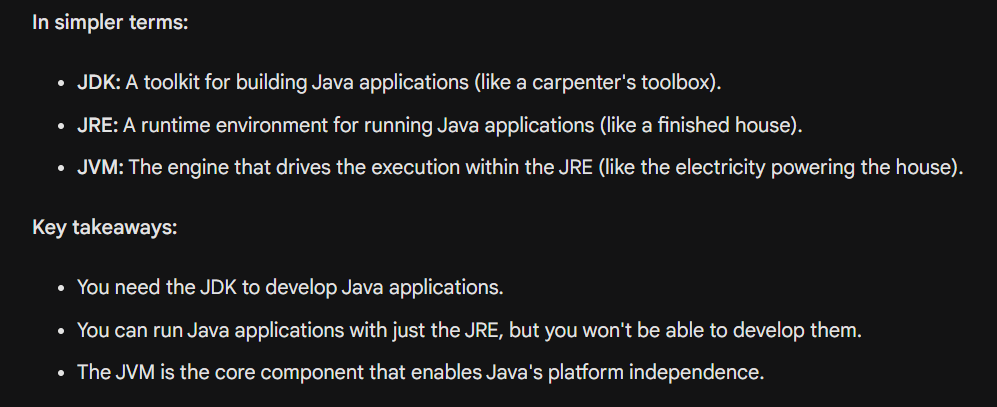


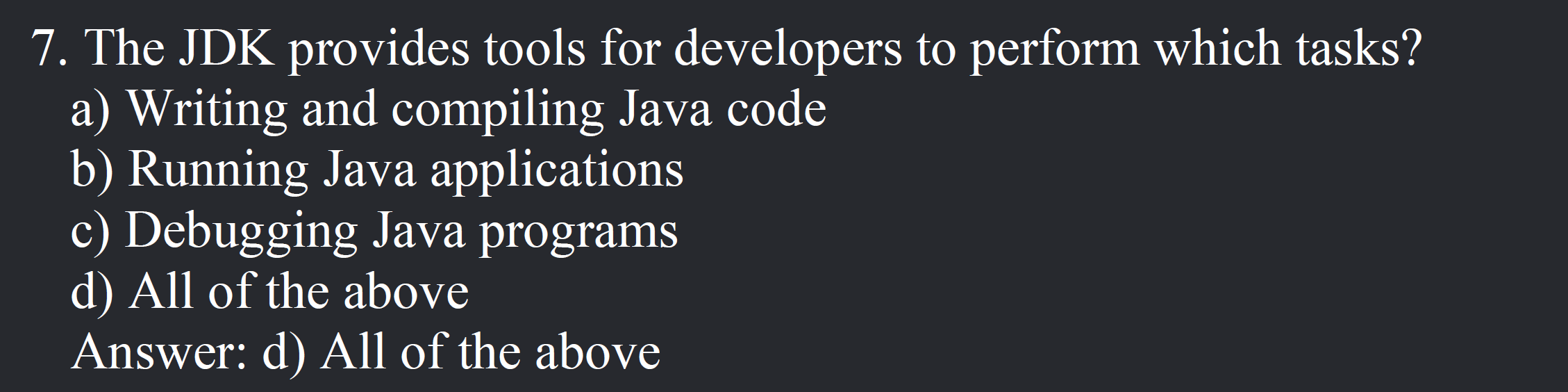


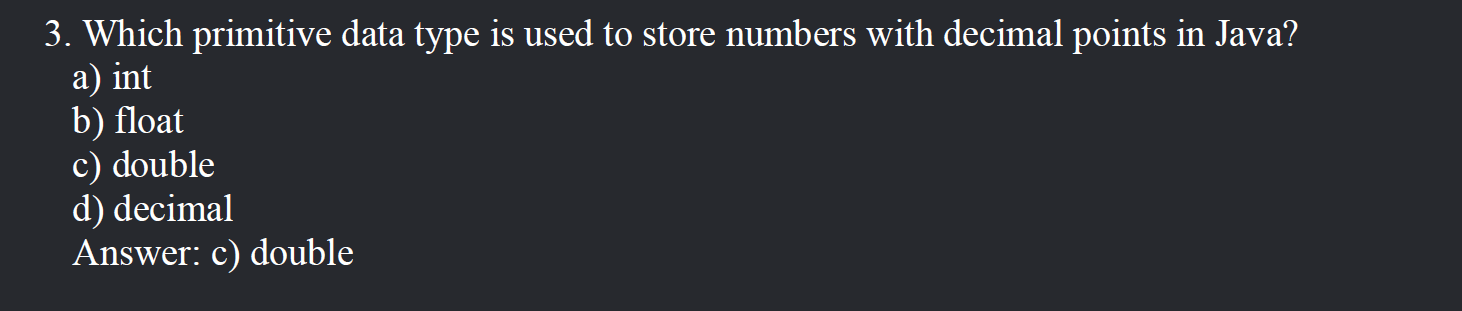


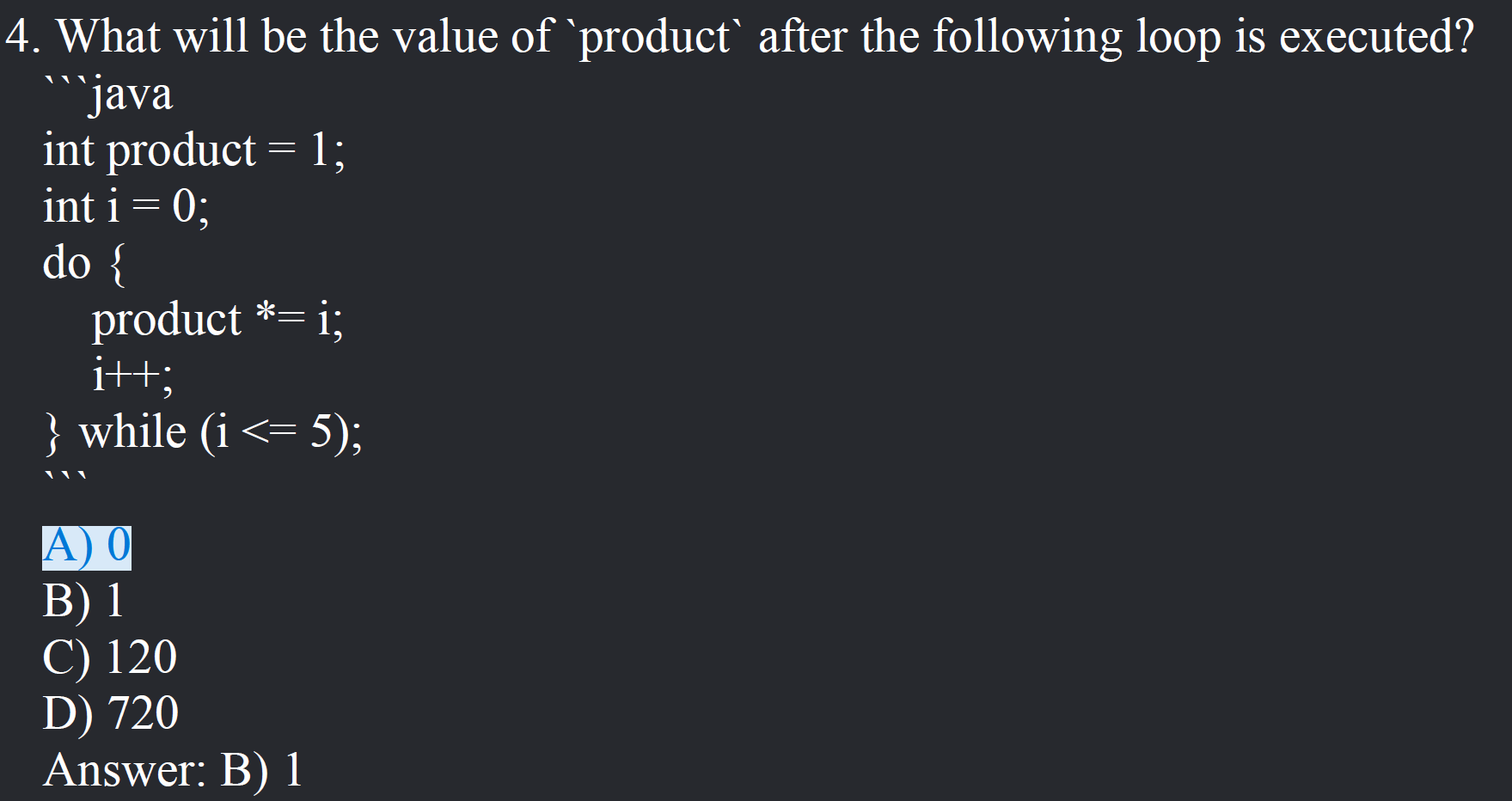


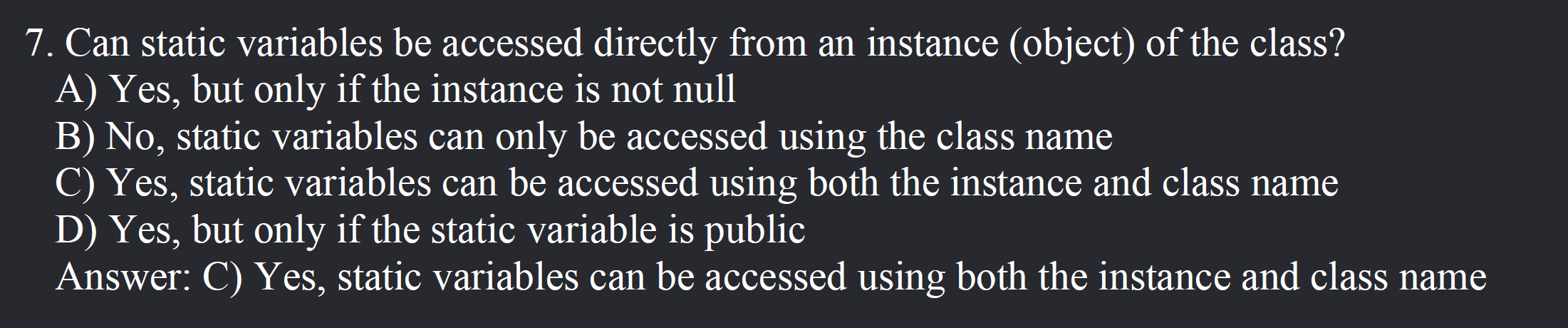


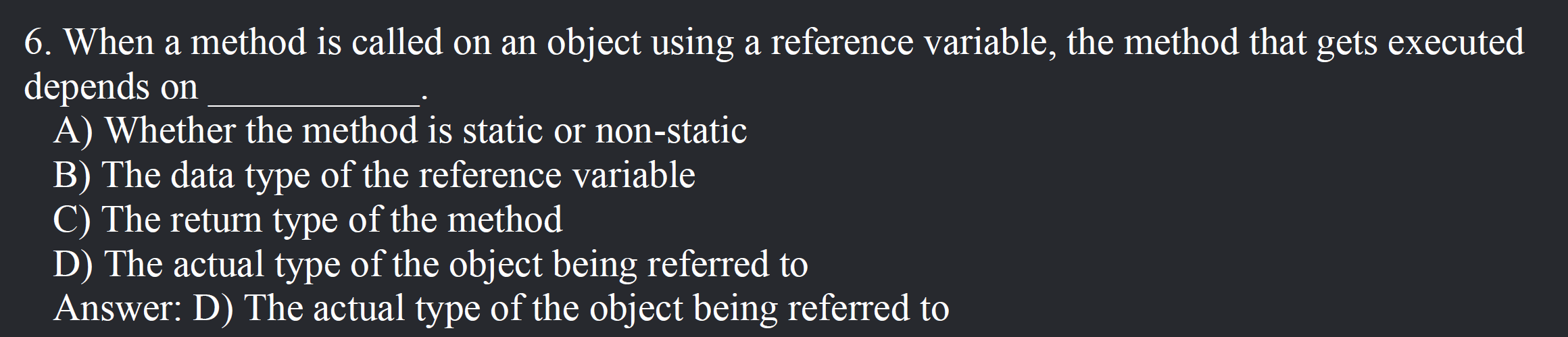


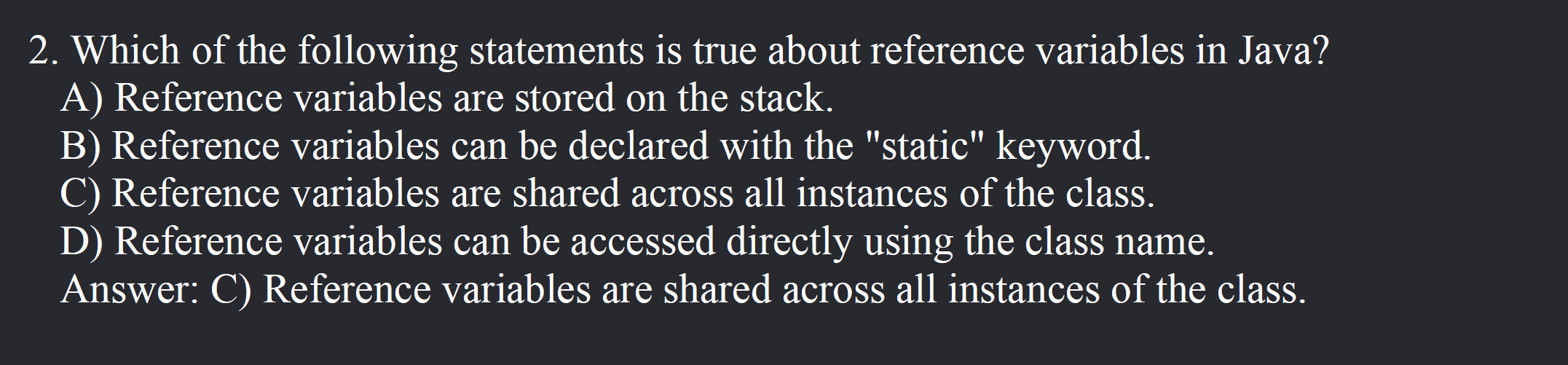


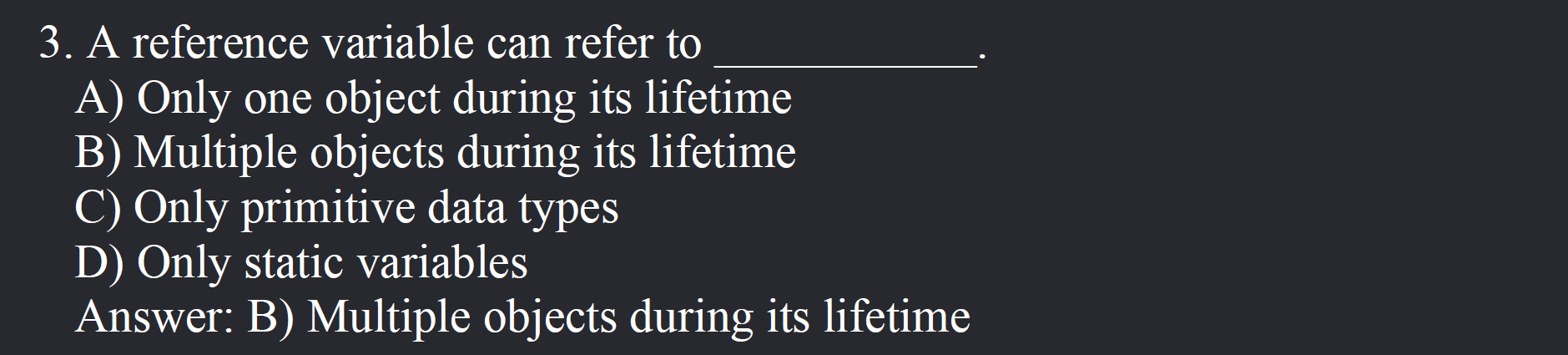


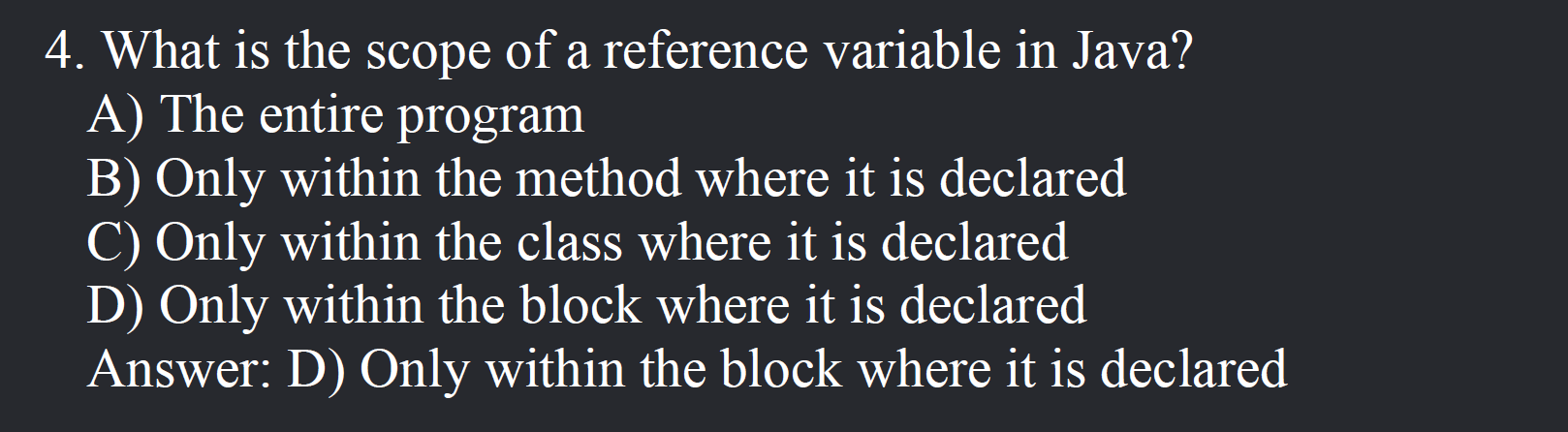


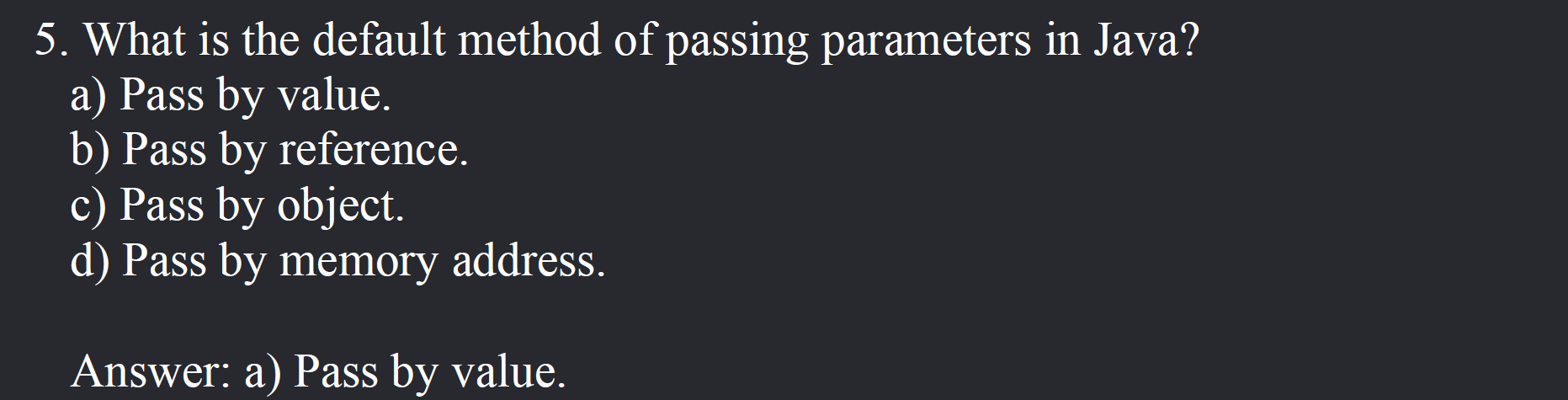


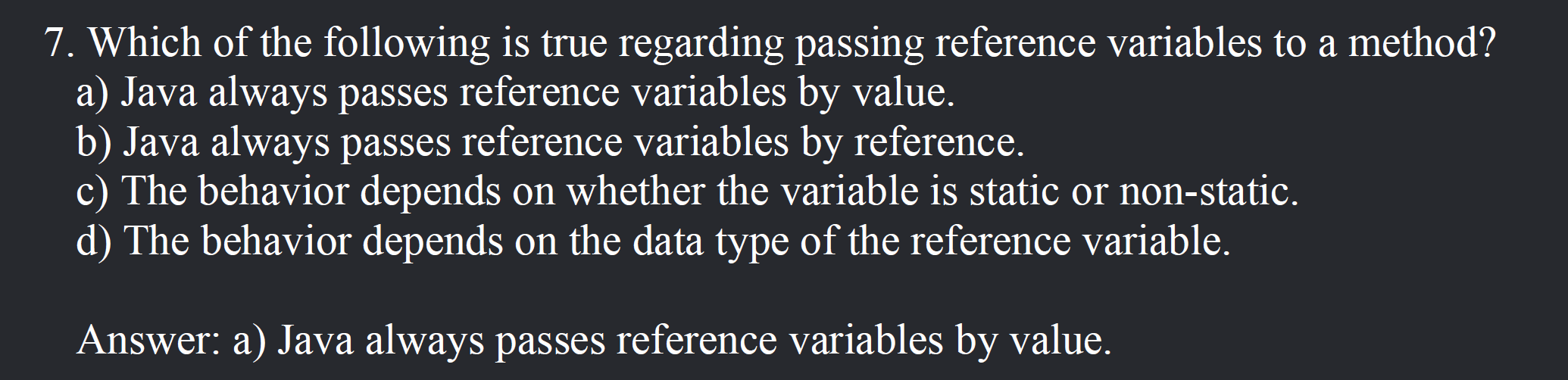


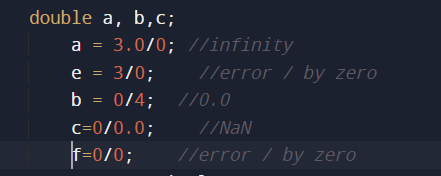
Starting from page 19 – 7th mcq





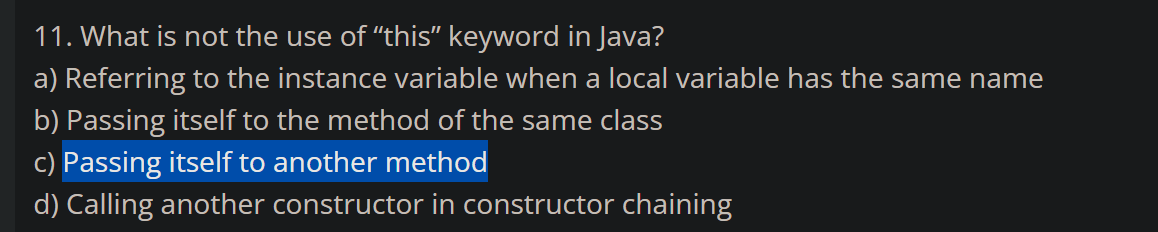


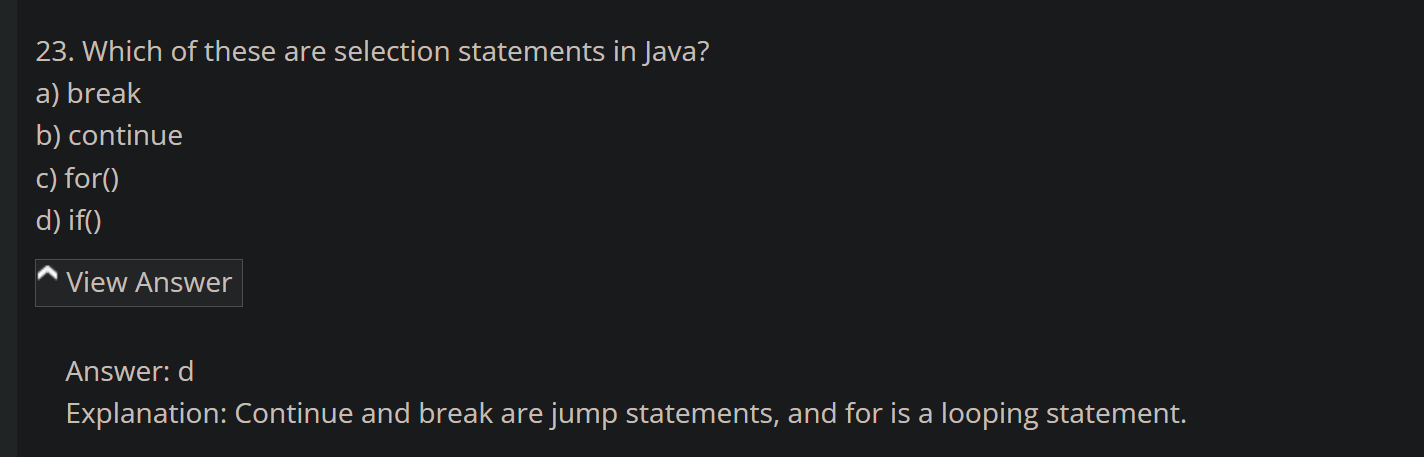




Sanfoundry mcq’s started from here

<https://www.sanfoundry.com/java-questions-answers-freshers-experienced/#java-chapters>





<https://www.sanfoundry.com/java-questions-answers-concepts-oops/>

