JAVA NOTES :

1. Java is Pass by Value for primitive types and pass by reference for objects.

1. Java code(.java file) → Byte code(.class file) → OS specific JVM → Machine code

2. JDK(Compiler + Debugger + JRE(Libraries + Components to run Java app +JVM(Virtual Machine that runs byte code)))

3. Java does not has multiple inheritance.

4. Wrapper Classes: Boolean(boolean),Byte(byte),Character(char),Double(double),Float(float),Integer(int),Long(long),Short(short).

5. Reasons why we need Wrapper Classes -null is a possible value -use it in a Collection.

6. valueOf Static Methods, provides another way of creating a Wrapper Object

Integer hundred = Integer.valueOf("100");//100 is stored in variable

Integer seven = Integer.valueOf("111", 2);//binary 111 is converted to 7

Example converting integer to its binary representation :

public class java\_rough {

public static void main(String[] args) {

int number = 4;

StringBuilder binaryRepresentation = new StringBuilder("");

while(number > 0) {

binaryRepresentation.append(String.valueOf(number & 1));

number = number >> 1;

}

binaryRepresentation = binaryRepresentation.reverse();

System.out.println(binaryRepresentation);

System.out.println(Integer.parseInt(binaryRepresentation.toString()));

}

}

\*\*Stringbuilder has .reverse() method, string does not.

\*\*String has toCharArraymethod, Stringbuilder does not.

7. The difference is that using the Constructor you will always create a new object, while using valueOf() static method, it may return you a cached value within a range.

8. Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and

their corresponding object wrapper classes.

9. Auto Boxing helps in saving memory by reusing already created Wrapper objects. Auto Boxing uses the static valueOf methods.

10. Value of a String Object once created cannot be modified. Any modification on a String object creates a new String object.

11. Objects of type String are immutable. StringBuffer is used to represent values that can be modified.

StringBuffer s3 = new StringBuffer("Value1");

12. Both String and StringBuffer are thread-safe. StringBuffer is implemented by using synchronized keyword on all methods.

13. Equals method is used when we compare two objects. Default implementation of equals method is defined in Object class. The implementation is similar to == operator. Two object references are equal only if they are pointing to the same object.

14. Super class reference variable can hold an object of sub class. And the methods for child class will be called.

15. Variables in an interface are always **public, static, final.** So while declaring any of the below can be used but variable will still be public, static, final.  
interface ExampleInterface1 {

//By default - public static final. No other modifier allowed

//value1,value2,value3,value4 all are - public static final

int value1 = 10;

public int value2 = 15;

public static int value3 = 20;

public static final int value4 = 25;

//private int value5 = 10;//COMPILER ERROR

}

Example of an interface with default and static methods definition in an interface:

public class java\_rough {

public static void main(String[] args) {

System.out.println(testingDefaultMethod.getSecondNumber(10));

testingDefaultMethod demo = new testingDefaultMethodImpl();

System.out.println(demo.getNumber(8));

}

public interface testingDefaultMethod {

default int getNumber(int number) {

return number;

}

static int getSecondNumber(int number) {

return number;

}

}

public static class testingDefaultMethodImpl implements testingDefaultMethod {

}

}

16. Interface methods are by default public and abstract.

17. An interface can extend another interface. The class implementing this interface must define methods for both child and parent interfaces, but **an interface cannot extend a class**.

interface SubInterface1 extends ExampleInterface1 {

void method3();

}

18. A class can implement multiple interfaces. It should implement all the method declared in all Interfaces

being implemented. Interface can have static and default methods defined in them from Java 8

Example for static and default methods being defined in an interface :

public class java\_rough {

public static void main(String[] args) {

System.out.println(testingDefaultMethod.getSecondNumber(10));

testingDefaultMethod demo = new testingDefaultMethodImpl();

System.out.println(demo.getNumber(8));

}

public interface testingDefaultMethod {

default int getNumber(int number) {

return number;

}

static int getSecondNumber(int number) {

return number;

}

}

public static class testingDefaultMethodImpl implements testingDefaultMethod {

}

}

19. An abstract class is a class that cannot be instantiated, but must be inherited from. An abstract class may be fully implemented, but is more usually partially implemented or not implemented at all, thereby encapsulating common functionality for inherited classes. If you want to provide common, implemented functionality among all implementations of your component, use an abstract class.

20. An Abstract method does not contain body. An abstract method does not have any implementation. The implementation of an abstract method should be provided in an over-riding method in a sub class. Abstract method can be declared only in Abstract Class, that can contain 0 or more abstract methods.

abstract void abstractMethod1();

21. Syntactical Differences between Interfaces and Abstract classes

• Methods and members of an abstract class can have any visibility. All methods of an interface must be public.

• A concrete child class of an Abstract Class must define all the abstract methods. An Abstract child class can have abstract methods. An interface extending another interface need not provide default implementation for methods inherited from the parent interface.

• A child class can only extend a single class. An interface can extend multiple interfaces. A class can implement multiple interfaces.

• A child class can define abstract methods with the same or less restrictive visibility, whereas a class implementing an interface must define all interface methods as public.

22. A constructor can call the constructor of a super class using the super() method call. Only constraint is that it should be the first statement. If a super class constructor is not explicitly called from a sub class constructor, super class (no argument) constructor is automatically invoked (as first line) from a sub class constructor.

23. Another constructor in the same class can be invoked from a constructor, using this({parameters}) method call. A constructor cannot be explicitly called from any method except another constructor.

public Animal() {

this("Default Name");

}

public Animal(String name) {

this.name = name;

}

24. If a super class constructor is not explicitly called from a sub class constructor, super class (no argument) constructor is automatically invoked (as first line) from a sub class constructor.

Order of invocation when default constructor without arguments prints its name.

Animal Constructor - Parent

Dog Constructor - Child

Labrador Constructor – GrandChild

25. **instanceof** operator checks if an object is of a particular type and can be used with interfaces as well.

**System.out.println(subClass2 instanceof Interface);//true**

26. Coupling is a measure of how much a class is dependent on other classes. There should minimal dependencies between classes. So, we should always aim for low coupling between classes.

27. Cohesion is a measure of how related the responsibilities of a class are. A class must be highly cohesive i.e. its responsibilities (methods) should be highly related to one another.

28. Common OOPs concepts: (For Symbols: https://javapapers.com/oops/association-aggregation-composition-abstraction-generalization-realization-dependency/)

1. Association: Two independent entities being related. EX: A Student and a Faculty are having an association.

2. Aggregation: Aggregation is a special case of association.It signifies one object having another object as its property.

For example consider two classes Student class and Address class. Every student has an address so the relationship between student and address is a Has-A relationship. But if you consider its vice versa then it would not make any sense as an Address doesn’t need to have a Student necessarily.

3. Composition: Composition is a special case of aggregation. It signifies one object having another object as its property but the inner object cannot exist without the main outer object. EX: Body: Heart

4. Generalization: Consider there exists a class named Person. A student is a person. A faculty is a person. Therefore here the relationship between student and person, similarly faculty and person is generalization.

5. Realization: The relationship between the interface and the implementing class.

**Encapsulation** is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction. Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java :-

- Declare the variables of a class as private.

- Provide public setter and getter methods to modify and view the variables values.

For Abstraction we can use abstract class/ interface inorder to generalize the methods involves and hiding there actual implementation which would be defined in the extending/implementing class.

29. A class declared directly inside another class and declared as static is a Static Inner Class. Static variables and methods are class level variables and methods. Static methods should be used when your function only depends on the input arguments and you don’t want it to be overriden. You can create an inner class inside a method. Anonymous Class does not have a name. Below are the use cases for these 3 classes types :-

- Anonymous Class → Declaring new Comparator inside a function and overriding directly overriding compare method.

Comparator<String> reverseComparator = new Comparator<String>() {

/\* Anonymous Class \*/

@Override

public int compare(String string1,

String string2) {

return string2.compareTo(string1);

}

};

- Inner static class – Increase encapsulation and its intialization can be done as OuterClass.StaticNestedClass nestedObject = new OuterClass.StaticNestedClass();

- Inner class - has access to all members(static and non-static variables and methods, including private) of its outer class and may refer to them directly in the same way that other non-static members of the outer class do, and its intialization can be done as OuterClass.InnerClass innerObject = outerObject.new InnerClass();

30. A class is called a Default Class is when there is no access modifier specified on a class. These classes are visible inside the same package only. The other access modifiers are public, private, protected.

31. Final class cannot be extended. Final methods cannot be overridden. Once initialized, the value of a final variable cannot be changed. Final arguments value cannot be modified inside the function.

32. Volatile can only be applied to instance variables. A volatile variable is one whose value is always written to and read from "main memory". Each thread has its own cache in Java. The volatile variable will not be stored on a Thread cache.

33. Default doesn't need to be the last case in an switch.

34. If we want some code to be always executed we can move it to finally block. Finally will be executed even when there is a return statement in try or catch. Try without catch and finally is not allowed, you should have one of them.

private static void method2() {

Connection connection = new Connection();

connection.open();

try {

// LOGIC

String str = null;

str.toString();

} catch (Exception e) {

// NOT PRINTING EXCEPTION TRACE - BAD PRACTICE

System.out.println("Exception Handled - Method 2");

} finally {

connection.close();

}

}

Output

Connection Opened

Exception Handled - Method 2

Connection Closed

Line after Exception - Method 1

Line after Exception – Main

35. Throwable is the highest level of Error Handling classes. Below class definitions show the pre-defined exception hierarchy in Java.

- class Error extends Throwable{} → Error is used in situations when there is nothing a programmer can do about an error. Ex:

StackOverflowError, OutOfMemoryError.

- class Exception extends Throwable{} → Exception is used when a programmer can handle the exception.

- class RuntimeException extends Exception{} → RuntimeException and classes that extend RuntimeException are called unchecked exceptions

36. Checked exceptions are checked at compile-time. It means if a method is throwing a checked exception then it should handle the exception using try-catch block or it should declare the exception using [throws keyword](https://beginnersbook.com/2013/04/java-throws/), otherwise the program will give a compilation error. Unchecked exceptions which are subtypes of Error and RuntimeException. Methods also don’t have to declare to throw unchecked exceptions. It’s because programs typically cannot be recovered from unchecked exceptions.

Checked Exception : nullpointer, arrayIndexOutOfBound, Arithematic

Unchecked Exception : fileNotFound, classNotFound, io, SQL

37. Serialization helps us to save and retrieve the state of an object. Serialization is to convert object state to some internal object representation(JSON). De-Serialization is to convert internal representation to object. To serialize an object it should implement Serializable interface.Transient variables are not serialized (transient int variable\_name, when de-serialization happens variable\_name value is set to default value i.e. 0). When a class is De-serialized, initialization (constructor’s, initializer’s) does not take place. The state of the object is retained as it is.

38. Java, we can only pass parameters by value.   
 1. Primitives, Strings(immutable): explicitly mark the return to change value in calling function else only local value will be updated.

2. Objects, StringBuilder(non – immutable): no need for marking return object explicitly in calling function.

39. BigDecimal and BigInteger examples:

import java.math.BigDecimal;

import java.math.BigInteger;

public class java\_rough {

public static void main(String[] args) {

BigInteger bigInteger = new BigInteger("200");

System.out.println(bigInteger.add(BigInteger.valueOf(500)));

System.out.println(bigInteger.subtract(BigInteger.valueOf(500)));

System.out.println(bigInteger.multiply(BigInteger.valueOf(500)));

System.out.println(bigInteger.divide(BigInteger.valueOf(500)));

BigDecimal bigDecimal = new BigDecimal("1000.01");

System.out.println(bigDecimal.add(BigDecimal.valueOf(100)));

System.out.println(bigDecimal.subtract(BigDecimal.valueOf(100)));

System.out.println(bigDecimal.multiply(BigDecimal.valueOf(100)));

System.out.println(bigDecimal.divide(BigDecimal.valueOf(100)));

}

}

40. Java dataTypes limits

byte 1 byte Stores whole numbers from -128 to 127

short 2 bytes Stores whole numbers from -32,768 to 32,767

int 4 bytes Stores whole numbers from -2,147,483,648 to 2,147,483,647

long 8 bytes Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

float 4 bytes Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits

double 8 bytes Stores fractional numbers. Sufficient for storing 15 decimal digits

boolean 1 bit Stores true or false values

char 1 byte Stores a single character/letter or ASCII values

41. Collections :

1. The add(int index, Object obj) method will throw : java.lang.IndexOutOfBoundsException if the index is greater than current size of the array.

EX :

import java.util.ArrayList;

import java.util.List;

class java\_rough {

public static void main(String[] args) {

List<Integer> numbers = new ArrayList<>();

numbers.add(1);

numbers.add(2,2);

numbers.add(10, 3);

numbers.add(5);

for(int i : numbers) {

System.out.println(i);

}

}

}

2. Sorting Arraylist by passing an Comparator implementation via variable and using subList as well

EX :

import java.util.ArrayList;

import java.util.Comparator;

import java.util.List;

class java\_rough {

public static class Person {

private int age;

private String name;

public Person(int age, String name) {

this.age = age;

this.name = name;

}

public static final Comparator<Person> SORT\_BY\_AGE\_FOLLOWED\_BY\_NAME = (firstPerson, secondPerson) -> {

int result = firstPerson.getAge() - secondPerson.getAge();

if(result == 0) {

result = firstPerson.getName().compareTo(secondPerson.getName());

}

return result;

};

public int getAge() {

return this.age;

}

public String getName() {

return this.name;

}

}

public static void main(String[] args) {

List<Person> members = new ArrayList<>();

members.add(new Person(14, "hannah"));

members.add(new Person(13, "hannah"));

members.add(new Person(14, "miley"));

members.add(new Person(13, "adam"));

members.add(new Person(12, "justin"));

members.add(new Person(16, "audrey"));

members.sort(Person.SORT\_BY\_AGE\_FOLLOWED\_BY\_NAME);

int splitPoint = getSplitPoint(members);

System.out.println("Children Below 14");

members.subList(0, splitPoint).forEach(person -> System.out.println(person.getName()));

System.out.println("Children ages 14 and Above");

members.subList(splitPoint, members.size()).forEach(person -> System.out.println(person.getName()));

}

private static int getSplitPoint(List<Person> memberSortedByAge) {

for(int i = 0; i < memberSortedByAge.size(); i++) {

if(memberSortedByAge.get(i).getAge() >= 14) {

return i;

}

}

return memberSortedByAge.size();

}

}

41. Common collections and associated methods are :-

- ArrayList → **add(object o), add(int index, object o)**, **get(int index)**, toArray(), .iterator(), **contains(Object o)**, **indexOf( Object o)**, l lastIndexOf( Object o), clear(), **remove(int index or Object o)**, removeRange(index i, index j)

- HashSet → **add(E e)**, clear(), **contains(Object o)**, **remove(Object o)**, iterator(), isEmpty(), size()

- TreeSet → void add(Object o), addAll(Collection c), clear(), contains(Object o), first(), Object last(), isEmpty(), Object clone(), size(), boolean remove(Object o), Iterator iterator(), Iterator iterator()

- LinkedList → **void add(int index, Object element), boolean add(Object o)**, boolean addAll(Collection c), boolean addAll(int index, Collection c), void addFirst(Object o), void addLast(Object o), **boolean contains(Object o)**, Object get(int index), Object getFirst(), int indexOf(Object o), int lastIndexOf(Object o), Object remove(int index), Object removeFirst/Last(), int size(), Object[] toArray()

- PriorityQueue → boolean add(object), boolean remove(object), Object poll(), Object peek() : Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty, Comparator comparator() : Returns the comparator used to order the elements in this queue, or null if this queue is sorted according to the natural ordering of its elements, boolean contains(Object o), Iterator iterator(), int size(), Object[] toArray()

- Iterator → boolean hasNext(), Object next()

42. Java I/O :

- A static intilization block can be for static properties of a non static class, which is used only once and updates the value of the static property.

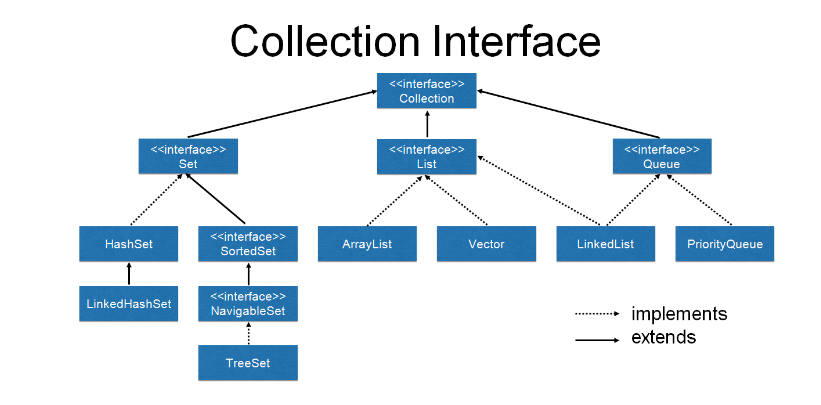
EX : private static String result;

static {

result = “settled”;

}

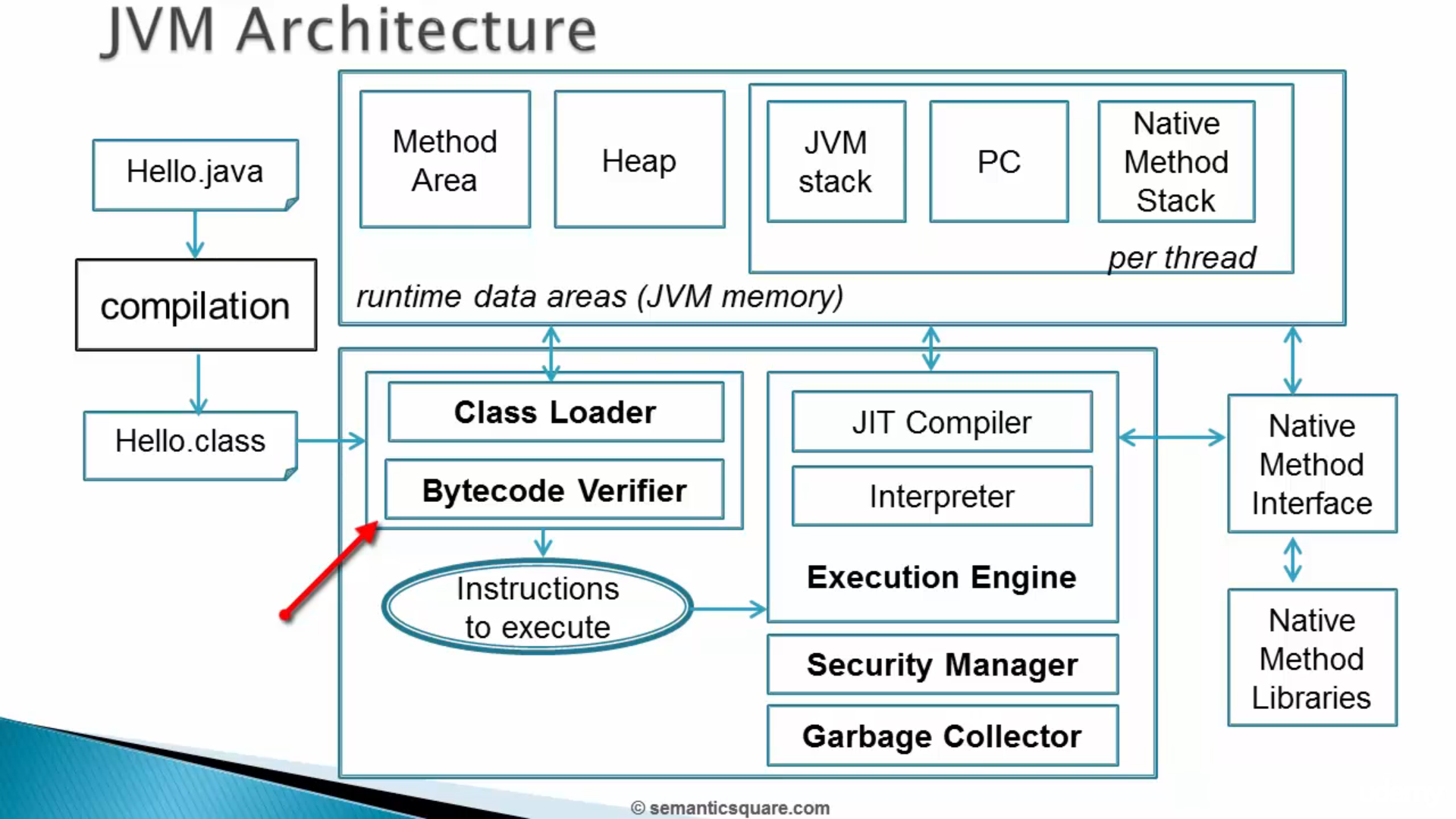
43. file.close() should be closed in finally block.



1. JVM interprets Java Byte Code which is faster than interpretation of Machine Language code.

2. JVM uses JIT which is Just in Time compilation where frequently used JBC is converted to MLC and cached.

3. Complete structure of program execution in Java



4. Common JAVA packages

- java.lang – used for fundametal classes. Ex – Strings {Imported by default}

- java.util - used for DS

- java.io - used for Reading and Writing

- java.sql - used for DB

- java.net - used for Networking

5. Accessing classes in same package can be done directly. For classes of other packages we need to use either of the 2 methods.

- Using import statements

- Using Fully Qualified Class Names {This approach is usually used when classes names are same in different packages so one of them can be used using Fully Qualified Class Names

EX – import java.util.Date followed by

Date date //util

java.sql.Date date2;

} .

6. Using \* import is not suggested as :-

- It may break your code if same class names are introduced in different packages using \* import.

- Code understanding is easier with explicit imports.

- It imports all classes in that package but not any sub-packages or their classes.

7. To store \*.class files associated with source code in a seprate directory you can use the below command.

- javac -d ../name\_of\_classes\_directory location/of/source/code.java

Here -d tells to create class files in the above mentioned directory hierarchy.

8. Naming conventions to follow for packages :-

- all lowercase alphabets

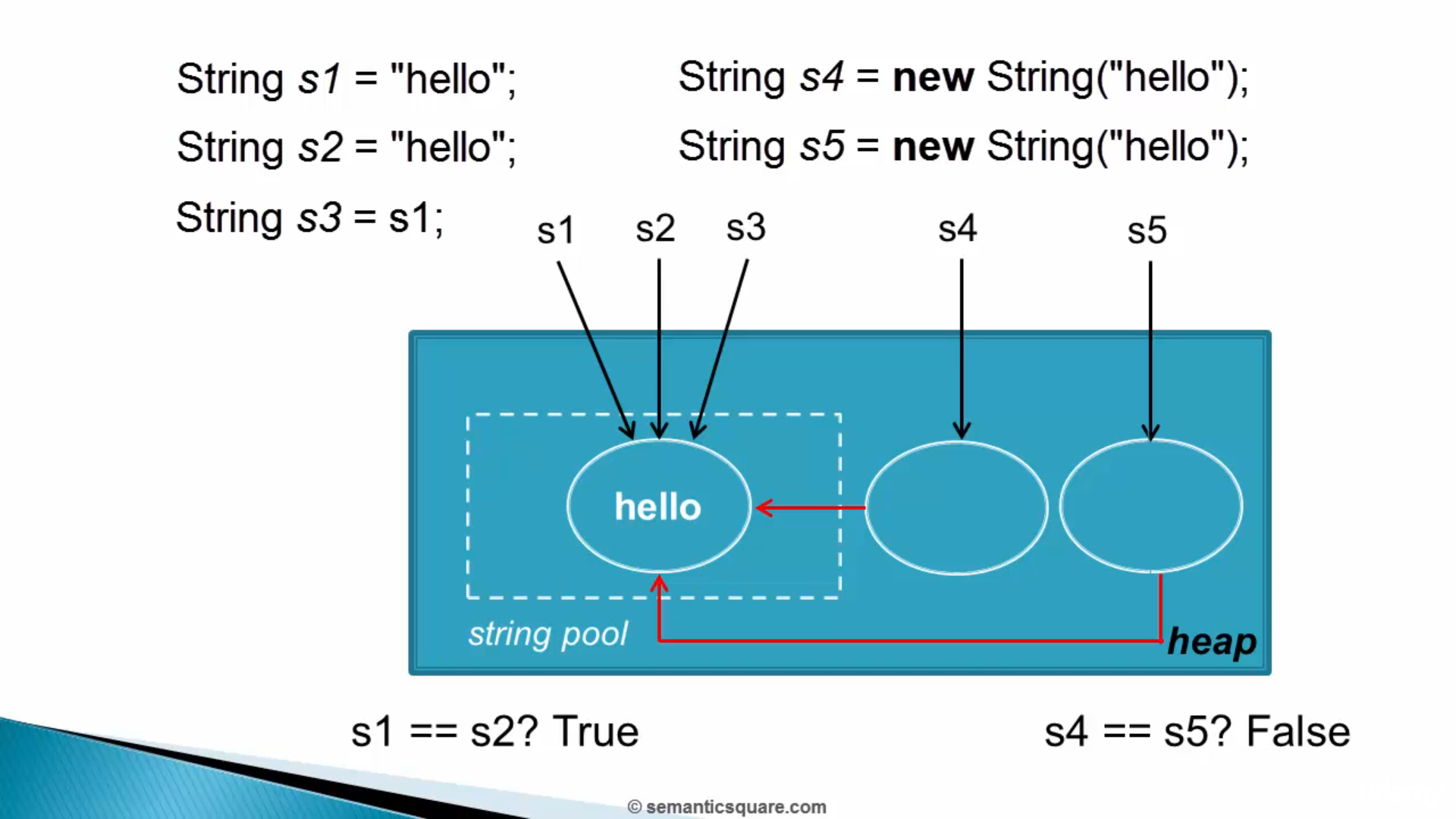
- acronyms can be used. EX – awt for Abstract window toolkit which is used in UI.

- Never start with java or javax as they are already used.

9. String Facts :-

- String is basically immutable sequence of unicode characters.

- String pool is area where strings which are created using literal and not new keyword are stored and can be resused to save memory.



If string pool doesn't contain the new literal both its reference in heap and its string object storage in pool will be created. Each element of a string pool is called an intern.

10. Common escape sequences :-

- double quote :\”

- single quote :\’

- backslash :\\

- new line :\n

- tab :\t

11. Some common design principals :-

- Using class variables as public causes problem of tight coupling of client and class code implementation. It can be solved by using private access modifier as :-

- Changes in implementation will not be passed on to user.

- User can only communicate using api’s and his data can be stored in a datatype other than what is received as he is not storing directly in variables.

- Using private constructor makes your class nonInstantiable.

- Using primitives is faster than using wrapper classes.

- Box classes don’t equate wrapped value but identities, but can be used with comparison operations like <,>,<=, >=.

- Use IS-A test to establish inheritance relations between classes. It is a one way test.

EX – Triangle IS-A Shape.

12. Because of polymorphism, what methods you can use for object of one class you can use for objects of its subclasses as well.

13. Final Variables – Cannot be modified.

Final Methods – Cannot be overriden.

Final Class - Cannot be Extended.

14. Also to avoid sub-classes from your class you can use private constructor to stop class intialization.

15. Interfaces don’t have state variables, but abstract class can have them.

16. Interfaces are used more often as a single class can implement multiple interfaces.

17. A skeletal class is an abstract class that has implemented a interface.

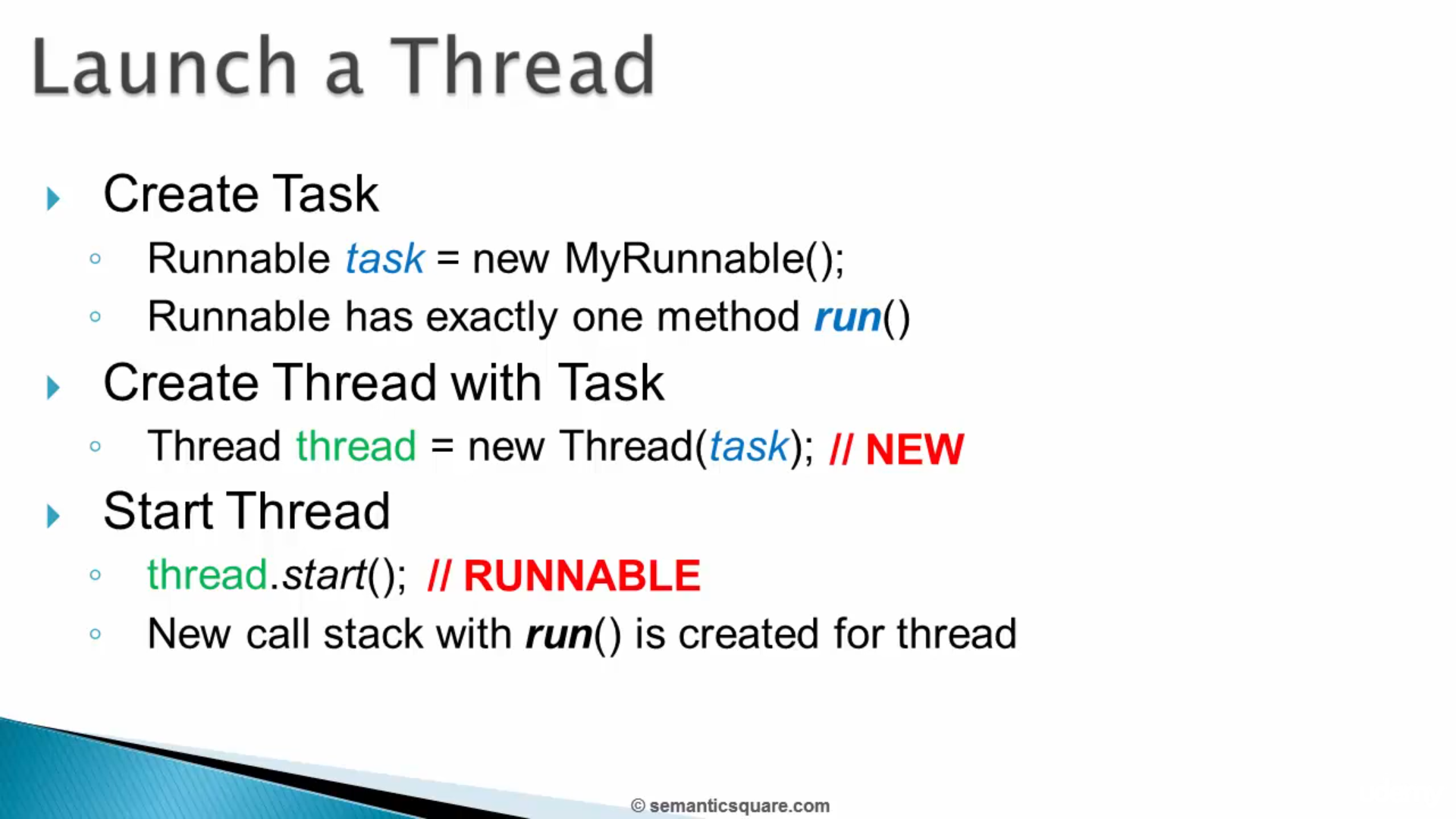
18. Threads are of 2 types :-

- Daemon – Background processes.

- Non-Daemon – Created within application to run service.

19. java.lang.Thread → represents the thread

java.lang.Runnable → represents the task



Doing the above we can create new non-daemon thread other than main thread. Once the thread is has finished execution you cannot use thread.start() again as the thread has now gone into dead state and will throw ‘illegal state’ exception.

EX : For storing time taken by process :

import java.util.List;

class java\_rough {

public static void main(String[] args) throws InterruptedException {

long startTime = System.currentTimeMillis();

System.out.println("Start of process : " + startTime);

Thread.sleep(10000);

System.out.println("Total time of process : " + ((System.currentTimeMillis() - startTime) / 1000));

}

}

20. ENUMS is a class storing only constants and its associated values. The Ordering priority associated with an ENUM constant within a class increases from left to right, which can be used to compare constants using the compareTo method.

EX:

class java\_rough {

public static enum ethinicity {

INDIAN("India"),

MEXICAN("Mexico"),

AMERICAN("USA");

private String country;

private ethinicity(String country) {

this.country = country;

}

public String getCountry() {

return this.country;

}

}

public static void main(String[] args) {

System.out.println(ethinicity.valueOf("INDIAN").getCountry());

}

}