**OCPJP SE 11**

**Table of contents**

**Building Block**

Calling Constructors

To create an instance of a class, all you have to do is write new before the class name with parentheses

Park p = new Park();

purpose of a constructor is to initialize fields,

public class Chicken {  
 int numEggs = 12; // initialize on line  
 String name;  
 public Chicken() {  
 name = "Duke"; // initialize in constructor  
 }  
}

Reading and Writing Member Fields

- It’s possible to read and write instance variables directly from the caller

public class Swan {  
 int numberEggs; // instance variable  
 public static void main(String[] args) {  
 Swan mother = new Swan();  
 mother.numberEggs = 1; // set variable  
 System.*out*.println(mother.numberEggs); // read variable  
 }  
}

public class Name {  
 String first = "Theodore";//writes to the fields  
 String last = "Moose";//writes to the fields  
 String full = first + last;//read & writes to the fields  
}

Executing Instance Initializer Blocks

**-code block ({})**

is code between the braces

**instance initializers**

is , code blocks appear outside a method

- they cannot exist inside of a method

How many blocks do you see ?

public class Bird {//class definition  
public static void main(String[] args) {//method declaration  
 { System.*out*.println("Feathers"); }//inner block  
 }  
{ System.*out*.println("Snowy"); }//instance initializer  
}

There are four code blocks.

balanced parentheses problem \*

- If there aren’t the same number of open ({) and close (}) braces

-if they aren’t defined in the proper order, the code doesn’t compile

**Following the Order of Initialization**

■ Fields and instance initializer blocks are run in the order in which they appear in the file.

■ The constructor runs after all fields and instance initializer blocks have run.

public class BuildingBlock {  
private String name = "Fluffy";  
{ System.*out*.println("setting field"); }  
public BuildingBlock() {  
 name = "Tiny";  
 System.*out*.println("setting constructor");  
 }  
public static void main(String[] args) {  
 BuildingBlock chick = new BuildingBlock();  
 System.*out*.println(chick.name);   
}  
}

*setting field*

*setting constructor*

*Tiny*

**Working with Data**

Packages

-package is a folder at the file system

-package declaration is done at the top one

package main;

public class Example1{

}

package main;//compile error

class A{}

package main;//compile error

class B{}

-don't put /avoid putting classes in the scr folder

-packages used to organises classes mainly that are related

-packages is a namespace where you're not allowed to declare classes of same name

Default package

-is the src package is a default package.

- you cannot have multiple public classes in the java file

-public tells that public class is the main class of the java file that can be called outside the other pages

public class name should be same with file name

-classes who are directly in the src folder does not have package declaration on the classes

-classes in different packages you should import the page

package-name.ClassName b= new package-name.Classname();

or

import package-name.ClassName;

import package-name.\*;//you can import all the classes

-package declaration should be in first line of a java file followed by classes import

Static import

public class B{

public static String a;

public static void m2(){

}

}

Main.java

import static second.B.a;

import static second.B.m;

import static second.B.m();//will not compile coz method has a brackets coz we mignt have diferent parameters on same method i.e polymophism

public class Main{

psvm(){

x=10;

m();

}

}

access modifiers

default access mode

-also called package

-if no access modified specified

-can be accessed within the package

-we are not allowed to use public,private,private and default access mondifer in the block of code this will throw an exception

private

-private variables or methods are only accessed whithin in the class

public

-

protected

accessed within a packages and also all subclasses

Order of access modifiers

public

protected //can be accessed outside the packages of the subclasses

default //only accessed within the package

private

**Reference type variable**

* piece of memory that stores the memory address where the object is located.

Date today = new Date();  
String greeting= new String("Hello kudzie, Good day");

Declaration = (instantiation) (Initialization)

* Object - value of reference type
* reference variables are stored in the stack
* objects are stored in the heap memory
* string is a special class
* String literals store in string pool

**Reference variables:**

* can be assigned to other objects or compatible type
* can be assigned to a new String object using new keyword
* can call methods
* they are of Classes
* can be assigned to null
* classes begin with uppercase letter e.g., String, Integer, Float

**Why we need instantiation in declaring reference variables coz we need space in the memory**

* Everything we do in java should be objects since java is an object-oriented programming

int i=5;//not acceptable in OOP

* We have to convert primitive data type to objects(wrappers) or vice versa

**Wrapper classes**

* corresponding to primitive types
* they are immutable
* they can hold a null value but primitive cannot hold a null value

Integer a1=null;//valid  
 int a2=null;//compile error

**Boxing**

* The process of putting/converting primitive value into the corresponding wrapper object
* Also called wrapping

int z = 5;  
 Integer ii = new Integer(15);  
Integer i2 = new Integer(z);  
System.*out*.println(ii);//15  
 System.*out*.println(i2);//5

**Autoboxing**

* start in java 5
* the compiler automatically converts a primitive to corresponding wrapper object
* Also called autowrapping

int i=34;  
Integer pounds = i;  
Integer pounds2 = 120;  
System.*out*.println(pounds);  
System.*out*.println(pounds2);

**Unboxing**

* Also called unwrapping
* Getting a primitive data from a corresponding wrapper object class

Integer ii=new Integer(33);  
int j =ii.intValue();  
System.*out*.println(j);

**Auto unboxing**

* the compiler automatically converts a wrapper classes to a primitive data type.

Integer value=20;//autoboxing  
int k=value;//auto unboxing  
char c = Character.*valueOf*('c');//the compilier will automatically converts reference varible into primitive  
System.*out*.println(c);

**NOTE**

* Primitive data type works faster than wrapper classes
* Certain framework in java works only with wrapper class i.e. generics, Streams API, Hibernate.

**Converting String to primitive data type**

String str="123";  
String strD='0.45';//compile error string wrapped in double quote but characters in into single quotes  
double d= Double.*parseDouble*(strD);  
int i= Integer.*parseInt*(str);

**Situation to use wrapper class**

* where variable cannot initialized with zero or any other value but can be initialized with null then we can use wrapper class
* used when it is required to use the primitive types as objects
* contain methods that unwrap the object and return the data type
* convert primitive data types into objects
* Collection framework such as Array List and Vector, Spring Data store only the objects (reference types) and not the primitive types.
* The object is needed to support synchronization in multithreading.

**converts a String into the wrapper class.**

int primitive = Integer.*parseInt*("123");  
Integer wrapper = Integer.*valueOf*("123");

Double apple = Double.*valueOf*("200.99");  
System.*out*.println(apple.byteValue()); // -56  
System.*out*.println(apple.intValue()); // 200  
System.*out*.println(apple.doubleValue()); // 200.99

**null**

* null reference can be assigned to any reference variables but not primitive variables
* calling any method on null gives a NullPointerException

String name=null;  
System.*out*.println(name.length());//throws NullPointerException

* a literal, a special constant you can point to whenever you wish to point to the absence of a value
* it is neither an object nor a type
* null equal null in Java is true

System.*out*.println(null==null);//true  
System.*out*.println(null.equals(null));//<nulltype> cannot be dereferenced

* a literal that represents a null reference
* Null Keyword refers to nothing
* Primitive data types cannot be null coz they store data directly

int b=null;//compile-time error

* Objects i.e referenced variables and Wrapper classes can be null coz they reference their data to the something into stack memory

ArrayList b=null;  
String a=null;  
var c=null;//compile error var cannot be null coz the compiler don't know the type of reference variable "c"  
if(a==null){  
 System.*out*.println("is null");//this will compile coz you’re comparing null with null i.e nothing to nothing  
}  
if(a.equals(null)){//will not compile results in NullPointerException coz you cannot call methods from null  
 System.*out*.println("is null");  
}  
if(b.isEmpty()){//will not compile you cannot call a method from null (nothing)  
 System.*out*.println("is null");  
}

NullPointerExeception

* Occurs when a reference object pointing to null then we try to call a methods

Cat myCat=null;  
myCat.getName();  
Boolean shouldPrintHello=null;//this compile coz Boolean is Wrapper class  
if(shouldPrintHello){  
 System.*out*.println("is null");  
}

**Strategy in avoiding NullPointerException**

* Always use primitive variables
* Initialize Wrapper Classes and References variables with empty variable

Cat cat = new Cat();  
List<String> list = new ArrayList<>();  
Stream<Integer> list2= Stream.*empty*();

* Always perform **NULL CHECK** before calling the method from an object

Cat cat = null;  
if(cat!=null){  
 System.*out*.println("is not null");  
}

* Try to avoid returning a null in the methods but return Optional or empty object or value

public Optional<Dog> getDogAgeByName(String name){  
 Dog dog = new Dog("Java",4);  
 return Optional.*ofNullable*(dog);  
}

* Just swap

String a=null;  
String str="Kudzie";  
if(a.equals(str)){//will not compile  
 System.*out*.print("is null equal");  
}

**Solution**

String a=null;  
String str="Kudzie";  
if(str.equals(a)){//will compile  
 System.*out*.print("is null equal");  
}

**Java Optional**

* used if we are not sure that method call will returns values or null
* Best way in java to handle a method that can return null or not
* Optional is the keyword that informs Java that the methods can have a value or not so the Java should be prepared in case the methods fail to return value

OptionTutorial.java

public class WithoutOptional {  
 public static void main(String[] args) {  
 Cat myCat=*findCatByName*("Kudzie");  
//you can handle null with a NULL CHECK or use of Optionals  
 if(myCat!=null){  
 System.*out*.println(myCat.getAge());//23  
 }else{  
 System.*out*.println(0);  
 }  
 }  
 public static Cat findCatByName(String name){  
//query from the database the age using the name  
 Cat cat = new Cat("Kudzie",23);  
 return cat;  
 }  
}

Cat.java

public class Cat {  
 private String name;  
 private int age;  
 public Cat(){}  
 public Cat(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }

//getters & setters  
 }

* the above program will run with problem if manage to get the values from the database but if we fail to get the values findCatByName() will return null which will cause NullPointerException

WithOptional.java

public class WithOptional {  
 public static void main(String[] args) {  
 Optional<Cat> optionalCat=*findCatByName*("Kudzie");  
//optionalCat.get() used to get the value which is inside the Optional Box  
//we haveto check if the Optional Box contains the object i.e NULL CHECK  
 if(optionalCat.isPresent()){  
 System.*out*.println(optionalCat.get().getAge());  
 }else{  
 System.*out*.println(0);  
 }  
 }  
 public static Optional<Cat> findCatByName(String name){  
//query from the database the age using the name  
 Cat cat = new Cat("Kudzie",23);  
 return Optional.*ofNullable*(cat);//putting the object into the Optional Box  
//if you know that the object you want to put in the Optional Box is not null you can use Optional.of(cat);  
 }  
}

Cat.java

public class Cat {  
 private String name;  
 private int age;  
 public Cat(){}  
 public Cat(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }

//getters & setters  
 }

**Use orElse()**

* this will return the passed parameter in the orElse() or to return the object in the OptionalBox

*Cat myCat= new optionalCat.orElse(new Cat("UNKNOWN",0));*

**Use orElseGet()**

* takes supplier as the parameter

**Use orElseThrow()**

* is the same method as .get() method

**optionalCat.map(Cat::getAge);**

* this will transform the Cat object in the otionalBox then we can get the .getAge() method

*optionalCat.map(cat -> cat.getAge())*

**Identifiers**

-names in java program e.g variable name, method name, class name, label name

Rules to define identifiers

- Identifiers must begin with a letter, a currency symbol, or a \_ symbol. Currency symbols include dollar ($), yuan (¥), euro (€),

-The allowed charaters in identifiers

a to z

A to Z

0 to 9

$,\_

2 identifiers can use digit but not at the start of the identifier names

3We cannot use keywords as identifiers

4. identifiers are case sensitive

5. All predefined class names and interface names can be used as identifiers (not recommended)

6. No length restriction

7 Two identifiers with same name not allowed

8 A single underscore \_ is not allowed as an identifier

9. You cannot use the same name as a Java reserved word

Which of the following are valid declaration in java ?

int int =10;//invalid  
int Int =10;//valid  
String string ="durga";//valid  
int #count=10;//invalid  
int 123total=100;//invalid  
class class =Test.class;//invalid  
int total=20;//valid  
int Runnable=10;//valid but not recommended  
String String ="kudzie";//valid but not recommended to use class names  
int \_total=30;//valid

public class String {//valid not recommended  
public static void main(java.lang.String[] args) {  
 System.*out*.println("String class");  
 }  
}

//Valid identifiers  
long okidentifier;  
float $OK2Identifier;  
boolean \_alsoOK1d3ntifi3r;  
char \_\_SStillOkbutKnotsonice$;

//These examples are not legal:  
int 3DPointClass; // identifiers cannot begin with a number  
byte hollywood@vine; // @ is not a letter, digit, $ or \_  
String \*$coffee; // \* is not a letter, digit, $ or \_  
double public; // public is a reserved word  
short \_; // a single underscore is not allowed

camelCase

- has the first letter of each word capitalized

- Method and variable names,

- Class and interface names are also written in camel case, with the first letter uppercase

snake\_case

**-** simply uses an underscore (\_) to separate words.

**-**uses uppercase snake case for constants and enum values, such as NUMBER\_FLAGS

literal

- When a number is present in the code

-Remember, you aren’t expected to memorize the maximum value for an int

-You won’t need to convert between number systems on the exam

long max = 3123456789; // DOES NOT COMPILE coz Java complains the number is out of range of int

Solution

long max2 = 3123456789L; // Now Java knows it is a long

long max3 = 3123456789l; // adding small L it looks like 1

decimal number system

- s numbering system is called base 10 since there are 10 possible values for each digit i.e 0-9

Literals and the Underscore Character

- you can have underscores in numbers to make them easier to read:

int million1 = 1000000;  
int million2 = 1\_000\_000;

You can add underscores anywhere except at the **beginning of a literal**, the **end of a literal**, **right before a decimal point**, **or right after a decimal point**.

double notAtStart = \_1000.00; // DOES NOT COMPILE  
double notAtEnd = 1000.00\_; // DOES NOT COMPILE  
double notByDecimal = 1000\_.00; // DOES NOT COMPILE

**-** You can even place multiple underscore characters next to each other, although we don’t recommend it.

double annoyingButLegal = 1\_00\_0.0\_0; // Ugly, but compiles  
System.*out*.println(annoyingButLegal);//1000.0  
double reallyUgly = 1\_\_\_\_\_\_\_\_\_\_2; // Also compiles  
System.*out*.println(reallyUgly);//12.0

Distinguishing between Primitives and Reference Types

|  |  |
| --- | --- |
| primitive types | Reference Types |
| primitive types have lowercase type names. | -starts with capital leter e.g Integer coz All classes that come with Java begin with uppercase |
| Primitives do not have methods declared on them | can be used to call methods, assuming the reference is not null. |
| Primitive types will give you a compiler error if you attempt to assign them null | reference types can be assigned null |
|  |  |

String reference = "hello";  
int len = reference.length();  
 int bad = len.length(); // DOES NOT COMPILE

int value = null; // DOES NOT COMPILE  
String name = null;

Using Primitive Types

**-** Java in two types of data: primitive types and reference types

- Java has eight built-in primitive data types

- eight primitive data types represent the building blocks for Java objects, because all Java objects are just a complex collection of these primitive data types.

- a primitive is not an object in Java, nor does it represent an object.

- primitive is just a single value in memory, such as a number or character.

Is String a Primitive?

No. String is an object class

**8 primitive Data types**

**Number Data Types**

**Non Number Data Types**

**Integral**

**byte**  -128 to 127  
**size** 1byte= 8bits

**short** -32768 to 32767  
 **size** 2bytes=16bits

**int** -2147483648 to

**size**:4bytes =32bits

**long** 2power63 to 2power63-1  
**size**:8bytes =64bits

**Floating point**  
  
float

double

boolean

char

|  |  |  |
| --- | --- | --- |
| Primitive type | Wrapper class | Example of intializing |
| boolean | Boolean | Boolean.valueOf(true) or new Boolean(true); |
| byte | Byte | Byte.valueOf((byte) 1) / new Byte(6); |
| stort | Short | Short.valueOf((short) 1) /new Short(3); |
| int | Integer | Integer.valueOf(1) /new Integer(4); |
| long | Long | Long.valueOf(1); /new Long(1567); |
| float | Float | Float.valueOf((float) 1.0) /new Float(3.6); |
| double | Double | Double.valueOf(1.0) /new Double(3.4); |
| char | Character | Character.valueOf('c') /new Character(‘A’); |

-The byte, short, int, and long types are used for integer values without decimal points.

- All of the numeric types are signed

- Each numeric type uses twice as many bits as the smaller similar type

- float requires the letter f or F following the number so Java knows it is a float

- Without an f or F, Java interprets a decimal value as a double

- long requires the letter l or L following the number so Java knows it is a long

- Without an l or L, Java interprets a number without a decimal point as an int

- Signed means it splits its range across the positive and negative integers eg

- char is unsigned, which means its range is strictly positive, including 0.

* we have to cast some value to correspond some data type

**Defining Text Blocks**

- text block starts and ends with three double quotes (""")

- the contents don’t need to be escaped.

String eyeTest = "\"Java Study Guide\"\n by Scott & Jeanne";

"Java Study Guide"

by Scott & Jeanne

String pyramid = """  
 \*  
 \* \*  
 \* \* \*  
 """;  
System.*out*.print(pyramid);

String block = """doe"""; // DOES NOT COMPILE coz s require a line break after the opening """  
String block2 = """  
 doe  
 """; // COMPILE

How many lines

Only one

String block3 = """  
doe \  
deer""";  
 System.*out*.println(block3);//doe deer coz coz the \ tells Java not to add a new line before deer.

String block4 = """  
doe \n  
deer  
""";  
 System.*out*.println(block4);// 4lines coz Since the text block has the closing """ on a separate line

String block5 = """  
"doe\"\"\"  
\"deer\"""  
""";  
 System.*out*.print("\*"+ block5 + "\*");

\*"doe"""  
 "deer"""  
 \*

escape characters

**Declaring Multiple Variables**

- You can declare many variables in the same declaration as long as they are all of the same type

- You can declare many variables in the same declaration as long as they are all of the same type

void sandFence() {  
 String s1, s2;  
 String s3 = "yes", s4 = "no";//there are 4 variables  
}

How many variables do you think are declared and initialized in the following code?

- only one of those values was initialized: i3.

void paintFence() {  
 int i1, i2, i3 = 0;  
}

int num, String value; // DOES NOT COMPILE coz tries to declare multiple variables of different types in the same statement

boolean b1, b2;//valid  
 String s1 = "1", s2;//valid  
 double d1, double d2;//invalid coz declare multiple variables in the same statement, they must share the same type declaration and not repeat it  
 int i1; int i2;//valid coz semicolon (;) separates statements in Java  
 int i3; i4;//invalid coz int i3 is a statement i4 is not declared  
 String s1, int i3;//invalid coz . Java does not allow you to declare two different types in the samestatement

**Initializing Variables**

local variable

-is a variable defined within a constructor, method, or initializer block

Final Local Variables

final int y = 10;  
int x = 20;  
y = x + 10; // DOES NOT COMPILE coz final variable cannot be reassigned

final int[] favoriteNumbers = new int[10];  
favoriteNumbers[0] = 10;  
//Initializing Variables  
 favoriteNumbers[1] = 20;  
 favoriteNumbers = null; // DOES NOT COMPILE coz final variable cannot be reassigned

**-** Local variables do not have a default value and must be initialized before use

- compiler will report an error if you try to read an uninitialized value

public int notValid() {  
 int y = 10;  
int x;//, x is not initialized before it is used

int reply = x + y; // DOES NOT COMPILE coz x is not initialized so cannot be used  
 return reply;  
 }

public void findAnswer(boolean check) {  
 int answer;  
 int otherAnswer;  
 int onlyOneBranch;  
 if (check) {  
 onlyOneBranch = 1;  
 answer = 1;  
 } else {  
 answer = 2;  
 }  
 System.*out*.println(answer);//if you remove one of answer intialization it will throw a compile error  
 System.*out*.println(onlyOneBranch); // DOES NOT COMPILE onlyOneBranch was not intialized before being used  
}

**Passing Constructor and Method Parameters**

constructor parameters

Variables passed to a constructor

method parameters,

Variables passed to a method

Constructor & method parameters are like local variables that have been preinitialized

rules for initializing constructor and method parameters are the same

public void findAnswer(boolean check) {}  
public void checkAnswer() {  
 boolean value;  
 findAnswer(value); // DOES NOT COMPILE coz value is not intialized  
}

**Defining Instance and Class Variables**

**instance variable**

-also called a field

-is a value defined within a specific instance of an object.

- require an instance to use

-**-** Instance variables do not require you to initialize them coz they are given a default value

**class variable**

-is one that is defined on the class level and shared among all instances of the class

- publicly accessible to classes outside the class and doesn’t require an instance to use.

- class variable uses keyword static before it

-**-** class variables do not require you to initialize them coz they are given a default value i.e

boolean b=false;  
Object obj=null;  
char ch= '\u0000';

**Converting from String**

*public static <<primitive>> parse<<Type>>(String s) throws NumberFormatException*

* returns primitive data type from a string

short a= Short.*parseShort*("1");  
int b = Integer.parseInt("234");

String str="0111";  
int t=Integer.*parseInt*(str);  
System.*out*.println(Integer.*parseInt*("-20"));//-20  
System.*out*.println(Integer.*parseInt*("+60"));//60  
System.*out*.println(t);

This is a common place on the exam that could result in a “Does not compile” answer

public static int **parse<<DataType>>(String s, int radix);** *throws NumberFormatException*

System.*out*.println(Integer.*parseInt*("20",16));//(2)\*16^1 + (0)\*16^0 = 32

[NumberFormatException](https://www.geeksforgeeks.org/numberformatexception-in-java-with-examples/) ocurs when :

1. *The string is null or of zero-length*
2. *The value represented by the string is not a value of type int*
3. *second argument radix is either smaller than Character.MIN\_RADIX or larger than Character.MAX\_RADIX.*
4. *If your String has leading zeroes, the parseInt() method will ignore them*

public static <<Wrapper>> valueOf(String s) *throws NumberFormatException*

The Integer.valueOf() method converts a String into an Integer object.

* returns a wrapper
* allows object catching

*Integer b=Integer.valueOf("2");*

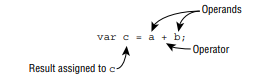
int decimalExample = Integer.*valueOf*("20");  
int signedPositiveExample = Integer.*valueOf*("+20");  
int signedNegativeExample = Integer.*valueOf*("-20");  
int radixExample = Integer.*valueOf*("20", 16);  
int stringExample = Integer.*valueOf*("geeks", 29);  
  
// Print statements  
System.*out*.println(decimalExample);//20  
System.*out*.println(signedPositiveExample);//20  
System.*out*.println(signedNegativeExample);//-20  
System.*out*.println(radixExample);//32  
System.*out*.println(stringExample);

**OPERATORS**

* special symbol applied to a set of variables, values or literals e.g +,\* - /
* Java operator is a special symbol that can be applied to a set of variables, values, or literals—referred to as operands— and that returns a result.

**Operand**

* value or variable the operator is being applied to 3+2
* refers to the value or variable the operator is being applied to



assignment operator (=) being used to store the result in variable c.

**Types of Operators**

**1. Unary**

* requires one operand

**2. Binary**

* takes two arguments

**3. Ternary**

* take three operands

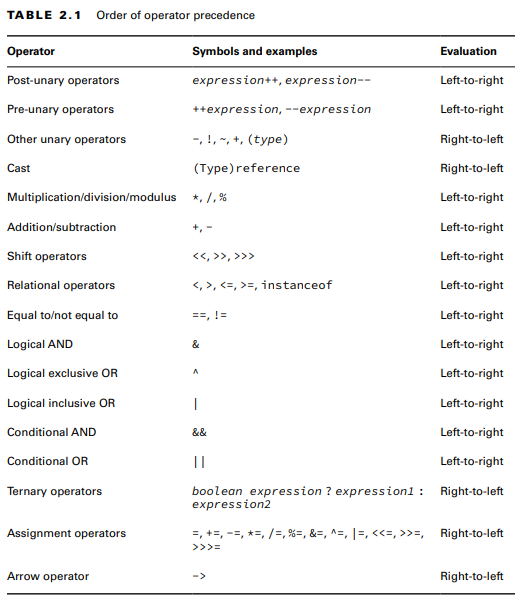
**Operator precedence ( THE RULES IN MATHEMATICS similar to body mass)**

* determine which operator to be evaluated in what order
* if two operators have same level of precedence then Java guarantees LEFT to RIGHT evaluation
* If there is parentheses() Java will simplify operations in the brackets first

int cookies = 4;  
 double reward = 3 + 2 \* --cookies;//3+2\*3 3+6=9 THEN PROMOTED TO DOUBLE =9.0  
System.out.print("Zoo animal receives: "+reward+" reward points"); // 9.0

|  |  |
| --- | --- |
| Post | Post |
| Pre | President of |
| U | United States |
| MD | My Dear |
| AS | Aunty Sally |
| SR EL STA | Senior EL Station |

|  |  |
| --- | --- |
| Operators | symbol and examples |
| Post-unary operators | exp++;exp-- |
| Pre-unary operators | ++exp;--exp |
| Other unary operators | -,!,+,~ (type) |
| Multiplication/division/modulus | \*,/,% |
| Addition/Substration | +, - |
| Shift operators | <<,>>,>>> |
| Relational operators | <,>,<=,>=, instanceof |
| Equalto/Not Equal to | ==,!= |
| Logical operators | &,^,| |
| Short circuit logical operators | &&, || |
| Ternary operators | boolean exp ? exp1 : exp2 |
| Assignement operators | =,+=,-=,\*=,/=,%=,&=,|=,>>=,<<=,>>>= |



* note that the order of precedence decreases as going down from POST to ASSIGNMENT
* for the exam we need to remember the Unary operators

|  |  |
| --- | --- |
| OPERATORS | Descriptions |
| ! | Inverts a boolean's logical |
| + | indicates a number is positive, although numbers are assumed to be the positive in java unless accompanied by a negative unary operator |
| - | indicates a literal is negative or negates any expression |
| ++ | increment a value by 1 |
| -- | decrement a value by 1 |
| (type) | cast a value to a specific type |

**NB**

* you cannot apply a negation operator (-) to a boolean expression, nor can you apply a logical complement separator (!) to a numerical expression.

boolean isanimalSleep = false;  
System.*out*.print(!isanimalSleep);//true  
System.*out*.print(!5) // does not compile

**INCREMENT & DECREMENT OPERATION**

**pre operation**

* We use ++i in our statement if we want to increment the value of i by 1 and then use it in our statement.
* ++i increment the value of i by 1 and uses the new value in the statement
* int parkAttendence=0;  
   System.*out*.println(++parkAttendence );//1  
   parkAttendence=0;  
   ++parkAttendence; //this will add 1 to the parkAttendence and return parkAttendence  
   System.*out*.println(parkAttendence);//1;  
    
   int i = 3;  
   int a=4;  
  int b = ++a; // b = 4, a = 4  
   for(int ii=0; ii<10;++ii){  
   System.*out*.println(ii);//print 0 to 9  
   }  
    
   int ia=1;  
   int j= ++ia +4;  
   System.*out*.println("i is"+ia+", j is"+j);//i is 2 , j is 6

**post operation**

* we use i++ in our statement if we want to use the current value, and then we want to increment the value of **i** by 1.
* i++ increments the value of i by 1 and uses the original value in the statement

int parkAttendence=0;  
 System.*out*.println(parkAttendence--);//--1  
  
 int i = 3;  
 int a = i++; // a = 3, i = 4  
  
 int parkAttendence2=0;  
 parkAttendence2--;//it first return the original value then decrement the value then  
 System.*out*.println(parkAttendence2);//-1  
  
 int lion =3;  
 int tiger = ++lion\* 5/ lion--; //4\*5/4 = 5 //lion value is 3 after the operation  
 int price = 2 \* 5 + 3 \*4 -8; //10 + 12 - 8 = 22 -8 =14  
//you can change the precedence of operation by wrapping it with brackets  
  
 int c=1;  
 int j= c++ +4;  
 System.*out*.println("i is"+i+", j is"+j);//i is 2 , j is 5  
  
 price = 2 \* ((5 + 3) \*4 -8);  
//2\*(8\* 4 - 8)  
//2\* (32 -8)  
// 2 \* 24  
// 48  
for(int ia=0; ia<10;ia++){  
 System.*out*.println(ia);//print 0 to 9

Range of byte is -127 to 127

byte b=127;  
b++;  
b++;  
b++;  
System.*out*.println(b);//-126 coz b++ goes out of range which is 128 it start the range with minimum value than to be promoted to int

NOTE

* Cannot apply the increment operator (++) on a constant value

int x = ++10;//will not compiles

* Pre-/post- increment & decrement does not promote values into next bigger data type it forces to start the range

byte c=127;  
c++;  
System.*out*.println(c);//-128  
  
var i=10;  
System.*out*.println(i++);//10  
System.*out*.println(i);//11  
int j=5,a=3,k=2,z=8;  
*m*(a++);//3  
*m*(j--);//5  
*m*(++k);//3  
*m*(--z);//7

void m(int i) {  
 System.*out*.println(i);  
}

**Reassignment**

int x=10;  
x=x++;//x was incremented to 11 but we use the original value of x which is 10  
System.*out*.println(x);//10  
int y=10;  
y=++y;  
System.*out*.println(y);//11  
  
int i=0;  
while(i<10){  
 System.*out*.println(i);//this will print an infinite loop of 0  
 i=i++;  
}  
  
int i1=0;  
while(i1<10){  
 System.*out*.println(i1);//0,1,2 ... 9  
 i1=++i1;  
}  
  
int x1=10;  
int y1=20;  
x=x1++ + ++x1 + --y1+ 2\*x1++;  
System.*out*.println(x);//10+12+19+2\*12=10+12+19+24=65

**NB**

* you cannot increment and decrement the final variable coz the final variables are constants
* the initialization block is executed once

final int x2=10;  
x2++;//will not compile  
5++;//will not compile

**Standard For Loop**

for(;;){//this compiles  
 System.*out*.println("print the for loop inifinite");  
}

**Same with the above for loop**

for(;true;){//this compiles  
 System.*out*.println("print the for loop inifinite");  
}

for(intialization block;condition block;control block){

}

for(int i=1;i<10;i++){  
 System.*out*.println(i);//print 1 2 3 ... 9  
}  
for(int i=1;i<10;++i){  
 System.*out*.println(i);//print 1 2 3 ... 9  
}

for(int i=0;i<=10;i++)  
 System.*out*.println("print the for loop "+i);//looped 0 to 10  
System.*out*.println("print the for loop inifinite");//runs once

for(var x=1;x<=10;x++){//can initialize with var  
 System.*out*.println(x);  
}

System.*out*.println(x);//compile error coz x can only be used in for loop only

int x=0;  
for(x=1;x<=10;x++){ //with var and don't declare x as int x=1;  
 System.*out*.println(x);//1 to 10  
}  
System.*out*.println(x);//11  
  
int y=0;  
for(y=10;y>=1;y--){  
 System.*out*.println(y);//10 to 1  
}  
System.*out*.println("0---"+y);//0

Initialization block omitted

int x2=10;  
for(;x2>=1;x2--){ //if you initialize & declare the variable  
 System.*out*.println(x2);//10 to 1  
}  
System.*out*.println("0---"+x2);//0

Controlling a for loop inside the for-loop

int x3=20;  
for(;;){  
 System.*out*.println(x3);//20 19 18 17 ... 6  
 x3--;  
 if(x3==5){  
 break;  
 }  
}

**Multiple for loop**

* separate multiple initialization blocks with a comma
* separate the condition block with && (and) || (or)
* separate multiple control blocks with a comma
* we not allowed to use var key word to declare multiple values i.e

for(var x=0,y=10;x<y; x++,y--){}//will not compile

for(int x4=0,y4=10;x4<y4; x4++,y4--){  
 System.*out*.println(x4+y4);//print 10 for five times  
}

**NB**

* initialization , condition and control block are optional
* int x5=10;  
  for(;x5>=1;){  
   System.*out*.println(x5);//10 to 1  
   x5--;  
  }  
  System.*out*.println("0---"+x5);//0

**Binary arithmetic operators**

|  |  |
| --- | --- |
| Operators | Description |
| + | Adds two numerial values |
| - | Subtracts two numerial values |
| \* | Multiples two numerial values |
| / | Divides one numerial value by another get the whole number |
| % | returns the reminder after the division of one numeric |

int num1=7;  
int num2=5;  
double b =num1/num2;  
System.*out*.println(b);//1.0 is promoted to double automatically if you want reminder use modulus

System.*out*.println("Modulus/Reminder");  
System.*out*.println(-12%5);//-2  
System.*out*.println(-11%5);//-1

**Parentheses ()**

* you can change the order of operation explicitly by wrapping parenthesis “()” around the sessions you want to evaluate first
* the parenthesis must be balanced. All left - parenthesis must be closed by right
* parenthesis before the end of the expression otherwise it will not compile

**Numerical type promotion**

* the conversion of a smaller numeric type to a larger numeric type, so that integer and floating-point operations may take place
* byte, char, and short values are converted to int values
* if two different data types, automatically promote to a larger data
* if integral and floating-point, automatically promote the integral value to the value floating-point
* Smaller data types, namely byte, short and char are first promoted to int before binary arithmetic operation. unary operators are operators are excluded e.g ++ to a short value result in short value
* After promotion the resulting value data type as its promoted operands

int x =1;  
 long y =33;  
 var z= x +y;  
// z is of long data type coz x is promoted to a bigger data type  
  
 double x2=39.21;  
 float y2 = 2.1F;  
 var z2 = x2 + y2;  
//z2 is of float data type coz x is promoted to a bigger data type which is float  
  
 short x3 =10;  
 short y3 = 3;  
 var z3 = x \* y;  
//z is of int data type coz short is promoted to int  
  
 byte a4=10;  
 byte b4=30;  
 int result= a4\*b4;//the result is now out byte range (-128 to 127) this result is automatically promoted  
 System.*out*.println(result);//300

* pre or post increment of short, char and byte doesn't promote the variable to int that is why

byte b3=127;  
b3++;//127  
b3++;//-128  
b3++;//-127  
b3++;//-126  
System.*out*.println(b3);//-125

short p=14;  
float q=13;  
double r =30;  
var j= r\*p/q;  
System.*out*.println(j); //z is of double value =32.30769230769231

Assignment operators =

-simple assignment operators

- used to assign the value on the right to the variable on the left

long kuu= 10;//kuu is assigned 5

-result of an assignment is an expression in and of itself, equal to the value of assignment

int wolf=0;  
long coyote = (wolf =3);  
System.*out*.println(coyote); // is 3 coz coyote is general assigned to wolf

boolean healthy = true;  
if(healthy = true){ // healthy is reassigned to true from false  
 System.*out*.println("Good !");  
}  
if(healthy = true)  
 System.*out*.println("Good !");

**CONVERSION**

-storing a smaller data type into a bigger data type

-this can be done implicitly and automatically converted i.e small data type to bigger data type

byte b=17;  
int c=12;  
System.*out*.println(c=b);//compiles coz small data type(byte) can fit into the big data type (int)

**CASTING (TYPE)**

-explicitly storing a variable into another variable i.e storing big data type into small data type but the big data type should fit into the range of the small data type

System.*out*.println("converting big data type but should be in range of the smaller to smaller datatype ");  
byte b=127;  
int a=1201;  
int c=13;  
//b=a;//cannot convert large type to small type  
b= (byte) a;//coz 12 can fit into the byte range (-128 to 12  
System.*out*.println(b);//-79= 1201%range of byte  
b=(byte) c;  
System.*out*.println(b);//13

System.*out*.println(1201%(127-(-128)));//181

- is a unary data type where one data type is explicitly interpreted as another

-optional and unnecessary when counting converting to a larger or widening data type

but it is required when converting to a smaller or narrowing data type of converting point number to an integral value

-without casting the compile will generate an error when trying to put a larger data type inside a smaller one

long c2 =120;  
int a2= (int) c;  
int fish = (int)1.0;//output 1  
short bird = (short) 1921222;// bird =20678  
int m= (int) 9f;//output 9  
int b2=5.6;//will not compile

**Solution**

float f=127.8f;//if you don’t put the f it will not compile  
int b3=(int) f;  
System.*out*.println(b3);//127 it will loss only point values

NB

You cannot cast or convert string to numerical data type or vice versa

String str="7";  
int b =(int) str;//will not compile string cannot be casted  
System.*out*.println(b);  
int i=7;  
String str2 =(String) i; //will not compile cannot convert int to String using casting  
System.*out*.println(str2);  
  
int a = 257;  
byte k=(byte) a;  
System.*out*.println(k);//1 = 257%range of byte

System.*out*.println(k%(127-(-128)));

**OVERFLOW**

-is when a number is so large that it cannot fit within a data type so the system " wraps around" to the lowest negative value and counts up from there

System.*out*.println("Overflow");  
System.*out*.println(Integer.*MIN\_VALUE*+1);//-2147483647

**UNDERFLOW**

-when the number is too low to fit the data type

byte b33= -200;//(-128 to 127)  
int fur = (int) 5; //this compiles but it was unnecessary to put (int)  
int fur = 5;  
int hair =(short) 2; //compile since java will automatically converts short into int  
int hair =2;  
String type = (String) "Bird";//compile but it was unnecessary to put (String)  
String type="Bird";  
long f= 10 (long);//not compile- double cannot go into float  
float egg = 2.0/9;//9 will convert to double but the result after diving will be double so double cannot be stored in float; so this will not compile  
int tadpole =(int)5 \* 2L;//will not compile coz 5 will promoted to long but the result will Long then it cannot be stored into int  
short = 3 - 2.0;//will not compile  
int fish =1.0;//will not compile  
short bird =1921222;// will not compile coz the value is out of range   
int mammal=9f;//will not compile coz you cannot store float (bigger) into int (small) data type  
long rep =1563575375355365;//this will not compile

**Examples of overflow**

byte b1 =-200;//this cannot be stored coz it’s out of short range  
short x=10;  
short y=3;  
short z = x\*y;//this will not compile but  
short z = (short) x\*y;  
  
short z = 1 + (short) x\*y;  
//this (short) x\*y will convert to int but the result will be int which cannot be stored into short   
  
short z= 30;//this will compiles the short will be stored into byte if it is in that range (-128 to 127)  
byte b = (byte) 1;  
b= (short) 1;  
b=(int) 1l;  
b=(int) 1.2345f;  
b= (int) 1.256353;  
  
//it’s possible to cast Wrapper data type you should use specific data type  
  
Float f= 1.2f;  
f= (float) 27;

**Compound assignment operator**

|  |  |  |
| --- | --- | --- |
| operator | Description |  |
| += | adds the value on the right to a variable on the left and assign the sum to a variable | x+=y //x=x+y; |
| -= | subtract the value on the right to a variable on the left and assign the difference to the variable | x-=y; //x=x-y; |
| \*= | multiples the value on the right to a variable on the left and assign the product to the variable | x\*=y; //x=x\*y; |
| /= | divide the value on the left to a variable on the right and assign the quotient to the variable | x/=y; //x=x/y; |
| %= |  | x%=y; //x=x%y; |

**NOTE**

-the left side of the compound operator can be applied only to a variable that is already defined and cannot be used to declare a new variable

-it explicitly cast a value

*int c=2, g=3;*

*c=c\*g;*

//or

*c \*= g;*

**benefit of compound**

*long a=10;*

*int b=5;*

*b=b \* a;//will not compile coz you cannot store long into int*

**solution and benefit**

*b\*=a;//this will do automatic casting*

**EQUALITY OPERATORS**

|  |  |  |
| --- | --- | --- |
| **operators** | **Apply to primitives** | **Apply to objects** |
| == | returns true if two values represent the same value | returns true if two values reference the same object |
| != | returns true if two values represent the different value | returns true if two values reference the different object |

**1.** **Comparing two numerical or character primitive types**

if the numerical values are different data types the values are automatically promoted

sout(*5==5.0)//* true

since the left value or side is promoted to a double

**2.** **Comparing two boolean value**

*sout(true==true);*

**3. Comparing two objects, including null and String values**

*String name1="sam";*

*String name2="sam";*

*sout(name1==name2); // true*

-String store the string into string pool

*File monday =new File("schedule.txt");*

*File tuesday =new File("schedule.txt");*

*File wednesday = tuesday;*

*sout(monday== tuesday);//false coz monday == tuesday is referencing to different objects in memory*

*sout(tuesday==wednesday);//true coz objects are referencing from same object*

-Two references are equal if they point to same objects or if both points to null

*sout(null==null);//true*

*boolean mky= true==3*;*//== has a higher precedence this will not compile coz the true and 3 is of different data type*

*boolean ap=false != "Grape";//will not compile you can not assign and compare on the same variable*

**RELATIONAL OPERATORS**

|  |  |
| --- | --- |
| Operators | Description |
| < | returns true if the value on the left is strictly less than the value on the right |
| <= | returns true if the value on the left is strictly less than or equal the value on the right |
| > | returns true if the value on the left is strictly greater than the value on the right |
| >= | returns true if the value on the left is strictly greater than or equal the value on the right |
| a instanceof b | Return true if the reference that a points to is an instance of a class, subclass, or class that implements a particular interface as named in b |

**INSTANCEOF OPERATOR**

-check to determine whether Employee is actually a person and Science Teacher is actually a person

interface Person{}  
class Employee implements Person{}  
public class Science\_Teacher extends Employee{  
 psvm(String [] a){  
 Employee e = new Employee();  
 Science\_Teacher s= new Science\_Teacher();  
 sout(e instanceof Person);//true  
 sout(s instanceof Employee);//true  
 sout(s instanceof Person);//true  
 }  
}

- Object is the parent/ super class in java for all classes in java

public void openZoo(Number time){  
 if(time instanceof Integer){  
 System.*out*.println((Integer)time+"O clock"); //good practice into always cast the value to the instance of super class  
 }else{  
 System.*out*.println(time);  
 }  
}

*openZoo*(5);//5 O clock  
*openZoo*(5.0);// 5.0 coz is double not an instanceof Integer  
*openZoo*(5.2f); // 5.2 coz is float not an instanceof Integer null instanceof Object;  
*openZoo*(null); // null will not compile coz null is not a subclass of Object

**LOGICAL OPERATORS**

-boolean data types are referred to as logical operators

-for numerical data type are referred to bitwise operators

**operator Description**

& Logical AND is true only if both values are true

| Inclusive OR is TRUE if at least one of the values is true

^ Exclusive XOR is true only if one value is true and other is false

The logical truth tables for &, ! and ^

|  |  |  |
| --- | --- | --- |
| x & y | | |
| (AND) | | |
| # | y=true | y= false |
| x=true | true | false |
| x=false | false | false |

|  |  |  |
| --- | --- | --- |
| x | y | | |
| (INCLUSIVE OR) | | |
| # | y=true | y= false |
| x=true | true | true |
| x=false | true | false |

|  |  |  |
| --- | --- | --- |
| x ^ y | | |
| (EXCLUSIVE OR) | | |
| # | y=true | y= false |
| x=true | false | true |
| x=false | true | false |

Here are some tips to help you remember this table:

AND is only true if both operands are true;

Inclusive OR is only false if both operands are false

Exclusive OR is the only true if the operands are different

boolean eyesClosed = true;  
 boolean breathingSlowly = true;  
 boolean resting = eyesClosed | breathingSlowly;  
 boolean asleep = eyesClosed & breathingSlowly;  
 boolean awake = eyesClosed ^ breathingSlowly;  
System.out.println(resting); // true  
 System.out.println(asleep); // true  
 System.out.println(awake); // false

SHORT CIRCUT OPERATORS (conditional operators)

-nearly identical to the logical operators, & and |, except that the returns right side of expression may never be evaluated if the final result can be determined by the left side of expression

int hour = 10;  
boolean zooOpen = true || (hour < 4);  
System.*out*.println(zooOpen); // true it only evaluates the right side only

-Avoid a NullPointerException

-used to check for Null b4 performing an operation

Animal d =null;  
if(d!=null & d.getAge<5){//this will not compile d is null so you cant call ,getAge()  
 sout("hello£");  
 }

Solution is ShortCircuit

if(d!=null && d.getAge<5){//this will compile coz the first one false so it will not evaluate the right side  
 sout("hello£");  
 }

Checking for unperformed side effects

operator description

&& Short circuit AND is true only if both values are true. If the left side is false, then the right side will not be evaluated.

|| Short circuit OR is true only if at least one of the values is true. If the left side is true, then the right side will not be evaluated.

int hour=10;  
boolean zooOpen=true || (hour< 4);//this will return true coz Short circuit OR is true when at least one value is true so the (hour <4 ) is not evaluated

Be careful with short circuit question in the exam

int rabbit=6;  
boolean bunny= (rabbit>=6) || (++rabbit<=7); //true but the value of rabbit is still 6 coz the right side is not yet evaluated  
  
boolean bunny= (rabbit>=6) | (++rabbit<=7); //this will return true coz rabbit is pre incremented to 7 then right and left is evaluated to check at least one side should be true for this logical operator

Logic operators

&& (short circuit)

& (standard)

||(short circuit)

|(standard)

!(Not)

-in the real applications use the short circuit rather than standard operators i.e | and &

Not (!)

boolean b=false;  
boolean result= !b;  
sout(result);  
int x=3;  
int y=5;  
boolean res2=!(x<y);  
sout(rs2);//false

AND (&&)

-this the binary operator

it used on two operands

boolean b1=false;  
boolean b2=false;  
boolean result= b1&&b2;//false  
boolean result= b1&b2;//false  
sout(result);

boolean result= b1&&b2;//short circult means that if the b1 on the left is false java will not excute the right right but the standard will always execute the right and leftside expression  
boolean b1=true;  
boolean b2=false;  
boolean res1=b1 && m(1); //true && true =true //1  
boolean res2=b2 && m(2);//false && true =false i.e m(2) will not executed coz the left side is false   
boolean res3=b1 & m(3); //true && true =true //3  
boolean res4=b2 & m(4);//false && true =false //4  
boolean m(int i){  
 sout(i);  
 return true;  
}

-try to do one thing on the one line of the code so that you code can be maintainable so its better to use short-circult this according to SOLID principle i.e single responsbility

short circuit OR (||) and standard OR (|)

boolean b1=true;  
boolean b2=false;  
boolean res1=b1 && m(1);  
boolean res2=b2 && m(2);//2  
boolean res3=b1 & m(3);//3  
boolean res4=b2 & m(4);//4  
boolean m(int i){  
 sout(i);  
 return true;  
}

TERNARY OPERATOR

-is a short of if statement

-also called conditional operator

booleanExpression ? expression1: expression2;

-second and third expressions can have different data types

-checking for unperformed side effects

int x= 10;

int b = x>20 ? 30 : "kudzie";//this will not compile the string can not be stored into int

int s=1, z=1;

int s= z<10 ? s++ :z++;//this returns s as 1 coz the value of s (expression 1) is assigned before incremented

//but if we check later this value of s is incremented but of z is still 1 coz expression was not excuted

sout(s +"values"+ z)

int owl = 5;  
int food;  
if(owl < 2) {  
 food = 3;  
} else {  
 food = 4;  
}  
System.*out*.println(food); // 4  
int owl2 = 5;  
int food2 = owl2 < 2 ? 3 : 4;  
System.*out*.println(food2); // 4

Note that it is often helpful for readability to add parentheses around the expressions in ternary operations,

int food1 = owl < 4 ? owl > 2 ? 3 : 4 : 5;  
int food3 = (owl < 4 ? ((owl > 2) ? 3 : 4) : 5);

you should know that there is no requirement that second and third expressions in ternary operations have the same data types, although it does come into play when combined with the assignment operator.

int stripes = 7;  
System.*out*.print((stripes > 5) ? 21 : "Zebra");  
int animal = (stripes < 9) ? 3 : "Horse"; // DOES NOT COMPILE

Ternary Expression and Unperformed Side Effects

int sheep = 1;  
int zzz = 1;  
int sleep = zzz<10 ? sheep++ : zzz++;  
System.*out*.print(sheep + "," + zzz); // 2,1

int sheep2 = 1;  
int zzz2 = 1;  
int sleep2 = sheep2>=10 ? sheep2++ : zzz2++;  
System.*out*.print(sheep2 + "," + zzz2); // 1,2

**Classes again and the final modifier**

- don't mix final and static variable

**final keyword**

-makes a variable a constant

-you can have final method parameter

void m(final int x){  
 x=10; //will not compile coz we're not allowed to reassign final variable  
}  
final int x;  
x=10;  
x=20;//will not compiles  
final var x=10;//whenever you are using var you should assign so that var can know the type of variable

final instance variables should be assigned in the same line with declaration coz can create instances any time and the instance variable will exist once you create an instance

public class Dog{  
 final String name="java";  
 final int id;//compile error  
}  
 Dog d1=new Dog();  
 sout(d1.name);//java  
d1.name="Tom";//compile error but final local variables can be declared and reassigned on the next line;  
 int m(){  
final int i;//compiles  
i=200;  
final var y;//will not compile coz final variable with var declaration should be initialized on the same line of declaration  
y=300;  
final var c=47.8;  
return i;

}  
 sout(m());//200

Using the constructor to provide a value to the final variable

Bottle.java

public class Bottle{  
 final doble volume;  
 //provide the value the moment we create the instance so we can assign the value in the constructor  
 Bottle(double volume){  
 this.volume=volume;  
 }  
}  
//create multiple instances with different values i.e Bottle can have different volumes  
 Bottle b1= new Bottle(1);  
 Bottle b2= new Bottle(3);  
b2.volume=3;//compile error coz the variable volume cannot be reassigned it’s a variable

Static Keyword

-used to declare methods and variables

-can be used to declare variable in the class only not methods or block of code

void m2(){  
 static int b=12;//will not compile coz static declaration not allowed in the block of the code  
}

Foo.java

public class Foo{  
 int x;//x belongs to the instance  
 static int *y*;//this doesn't belong to the instances  
 final static int *z*=10;  
}

Main.java

Foo f1= new Foo();  
Foo f2= new Foo();  
f1.x=10;  
f2.x=20;  
System.out.println(f1.x);//10  
System.out.println(f2.x);//20

-x value can be different with instance

Accessing the static variable

Foo.y=12;  
System.out.println(Foo.y);//12  
f1.y=40;  
f2.y=50;  
System.out.println(Foo.y);//50  
System.out.println(f1.y);//50  
System.out.println(f2.y);//50

-y value can take the last value assigned to y this depends on class

Static Lock

final static int z;  
final int w;  
//static block  
static {  
 z=10;  
 }  
//none static block  
 {  
 w=50;  
 }

NB

static is used to detach the variable from the instance

-static cannot be declared in the block of code

-static can have same value for the class or on all instances

-static is linked to class called class variable

Final

-final instance variables need to be assigned on the same line of declaration but also you can declare it on the second line using a constructor

-final local variable can be assigned on the different lines of code

-can be declared in block of code

-final variables cannot be reassigned again

-used to declare constant

**switch statement**

**-** single value is evaluated and flow is redirected to the first matching branch

- If no such case statement is found that matches the value, an optional default statement will be called.

switch(variableToTest) {  
 case constantExpression1:  
 // Branch for case1  
 break;  
 case constantExpression2, constantExpression3  
 // Branch for case2 and case3  
 break;  
 ...  
 default:  
 // Branch for default  
}

**break**

-used to break the program execution and the forces the execution to go out of the

- terminates the switch statement and returns flow control to the enclosing process.

- are optional, but without them the code will execute every branch following a matching case statement, including any default statements it finds

public static void printSeason(int month) {  
 switch(month) {  
 case 1, 2, 3: System.*out*.print("Winter");  
 case 4, 5, 6: System.*out*.print("Spring");  
 default: System.*out*.print("Unknown");  
 case 7, 8, 9: System.*out*.print("Summer");  
 case 10, 11, 12: System.*out*.print("Fall");  
 } }

*printSeason*(2);// WinterSpringUnknownSummerFall

**switch**

-if there is no break on each case of the switch the all the cases without break will be executed

switch(month) {} //valid switch without case

int x=10;  
switch(x){  
 case 1:  
 System.*out*.println("A");  
 case 10:  
 System.*out*.println("B");//B  
 case 15:  
 System.*out*.println("C");//C  
}

case values are constant

-default case if for other values of the case that are not matching

-default case can be placed in any place in the switch

default:  
 System.out.println("default case");  
break;

Changes to java 12

Switch with cases of multiple value

**Combining case Values**

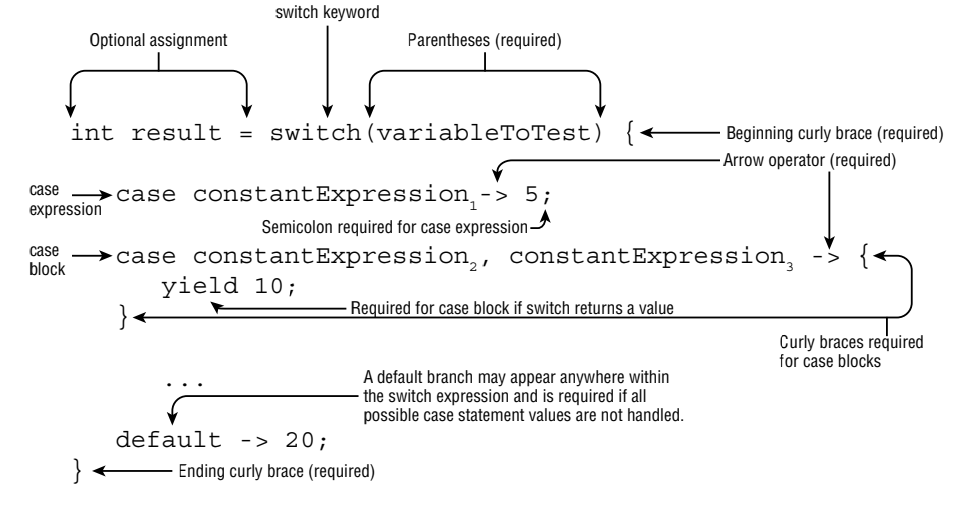
switch(animal) {  
 case 1,2: System.*out*.print("Lion");  
 case 3: System.*out*.print("Tiger");  
}

int y=10;  
switch(x){  
 case 10 -> System.*out*.println("value 10 "+y);//value 10 10  
 case 20,30,40, 50 -> System.*out*.println("value 20,30,40,50"+y);  
 default -> System.*out*.println("default value"+x);  
}

-No break

-the syntax is not verbose

Assing a switch statement / switch Expression



- all case and default branches must return a data type that is compatible with the assignment

- switch expression supports two types of branches: an expression and a block.

public void printDayOfWeek(int day) {  
 var result = switch(day) {  
 case 0 -> "Sunday";  
 case 1 -> "Monday";  
 case 2 -> "Tuesday";  
 case 3 -> "Wednesday";  
 case 4 -> "Thursday";  
 case 5 -> "Friday";  
 case 6 -> "Saturday";  
 default -> "Invalid value";  
 };  
 System.*out*.print(result);  
}

*printDayOfWeek*(5);//Friday

-a semicolon is required after each switch expression & cases

var result = switch(bear) {  
 case 30 -> "Grizzly"//not compiles coz the case has no semi colon  
 default -> "Panda"//not compiles coz the case has no semi colon  
}////not compiles coz the switch has no semi colon

Solution

var result = switch(bear) {  
 case 30 -> "Grizzly";  
 default -> "Panda";  
};

int bear;  
var result = switch(bear=30) {  
 case 30 -> "Grizzly";  
 default -> "Panda";  
};  
System.*out*.println(result);//Grizzly

Switch rules

1. All of the branches of a switch expression that do not throw an exception must return a consistent data type (if the switch expression returns a value).

2. If the switch expression returns a value, then every branch that isn’t an expression must yield a value.

3. A default branch is required unless all cases are covered or no value is returned.

Returning Consistent Data Types

int measurement = 10;  
int size = switch(measurement) {  
 case 5 -> 1;  
 case 10 -> (short)2;// short can be implicitly cast to an int  
 default -> 5;  
 case 20 -> "3"; // DOES NOT COMPILE coz *returns a type that cannot be assigned to the int variable*  
 case 40 -> 4L; // DOES NOT COMPILE  
 case 50 -> null; // DOES NOT COMPILE  
};

Applying a case Block

- switch expression supports both an expression and a block ({}) in the case and default branches.

- yield keyword is equivalent to a return statement within a switch expression

-yield used to avoid ambiguity about whether you meant to exit the block or method around the switch expression

int fish = 5;  
int length = 12;  
var name = switch(fish) {  
 case 1 -> "Goldfish";  
 case 2 -> {yield "Trout";}  
 case 3 -> {  
 if(length > 10) yield "Blobfish";  
 else yield "Green";  
 }  
 default -> "Swordfish";  
};

int fish2 = 5;  
int length2 = 12;  
var name2 = switch(fish2) {  
 case 1 -> "Goldfish";  
 case 2 -> {} // DOES NOT COMPILE coz t does not return a value using yield  
 case 3 -> {  
 if(length2 > 10) yield "Blobfish";  
 // yield "kudzie";  
 } // DOES NOT COMPILE coz other part of if statement don't returns all branches must yield a value within the case block.  
 default -> "Swordfish";  
};

Semicolons in switch Expressions

var name = switch(fish) {  
 case 1 -> "Goldfish" // DOES NOT COMPILE (missing semicolon)  
 case 2 -> {yield "Trout";}; // DOES NOT COMPILE (extra semicolon)  
} // DOES NOT COMPILE (missing semicolon)

Covering All Possible Values

- switch expression that returns a value must handle all possible input values

As a developer, there are two ways to address this:

■ Add a default branch is the most used option in java

■ If the switch expression takes an enum value, add a case branch for every possible enum value only applies only to switch expressions that take an enum

-that is why Even smaller types like byte are not permitted by the compiler, despite there being only 256 possible values.

- Since all possible permutations of Season are covered, a default branch is not required in this switch expression

enum Season {*WINTER*, *SPRING*, *SUMMER*, *FALL*}  
String getWeather(Season value) {  
 return switch(value) {  
 case WINTER -> "Cold";  
 case SPRING -> "Rainy";  
 case SUMMER -> "Hot";  
 case FALL -> "Warm";  
 };  
}

- consider including a default branch in every switch expression

int canis=3;  
String type = switch(canis) { // DOES NOT COMPILE coz thw switch does not cover all possible input values  
 case 1 -> "dog";  
 case 2 -> "wolf";  
 case 3 -> "coyote";  
};

public void printSeason(int month) {  
 switch(month) {  
 case 1, 2, 3 -> System.*out*.print("Winter");  
 case 4, 5, 6 -> System.*out*.print("Spring");  
 case 7, 8, 9 -> System.*out*.print("Summer");  
 case 10, 11, 12 -> System.*out*.print("Fall");  
 } }

int z=10;  
int b=switch(z){  
 case 10 -> (int) 2.5;//2  
 case 20,30,40, 50 -> (int)1.5;  
 default -> 300;  
  
};  
System.*out*.println("switch value"+b);//2

switch that does not compile

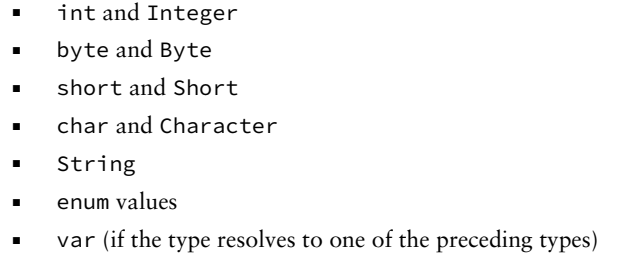
int month = 5;  
switch month { // DOES NOT COMPILE because it is missing parentheses around the switch variable  
 case 1: System.*out*.print("January");  
}  
switch(month) // DOES NOT COMPILE coz it is missing braces around the switch body  
case 1: System.*out*.print("January");  
switch(month) {  
 case 1: 2: System.*out*.print("January"); // DOES NOT COMPILE coz a comma (,) should be used to separate combined case statements, not a colon (:).  
}

-it might make more sense to throw an exception in the default branch if no match is found.

Selecting switch Data Types

- switch statement has a target variable that is not evaluated until runtime.

- target can include select primitive data types (int, byte, short, char) and their associated wrapper classes (Integer, Byte, Short, Character).



- enum, represents a fixed set of constants

-Boolean, Long, Float, and Double classes of Wrapper classes & associated boolean, long, float, and double of primitive data types are excluded in the switch statement

-the reason why boolean having too small a range of values and floating-point numbers having quite a wide range of values.

**Determining Acceptable Case Values**

- any variable or value can be used in a case statement

- values in each case statement must be compile-time constant values of the same data type as the switch value

final int getCookies() { return 4; }  
void feedAnimals() {  
 final int bananas = 1;  
 int apples = 2;  
 int numberOfAnimals = 3;  
 final int cookies = getCookies();  
 switch(numberOfAnimals) {  
 case bananas: //value is known at compile-time so its valid.  
 case apples: // DOES NOT COMPILE not permitted coz the varible is not a constant  
 case getCookies(): // DOES NOT COMPILE coz methods are not evaluated until runtime  
 case cookies : // DOES NOT COMPILE coz methods are not evaluated until runtime  
 case 3 \* 5 : //compiles coz 3\*5 value can be resolved at compile-time  
 } }

-cases values should fit the switch data type without an explicit cast

**if Statement**

**Verifying That the if Statement Evaluates to a Boolean Expression**

int hourOfDay = 1;  
if(hourOfDay) { // DOES NOT COMPILE  
}

**Shortening Code with Pattern Matching**

Pattern matching

- technique of controlling program flow that only executes a section of code that meets certain criteria

- reduce boilerplate in your code.

-can be used in switch & instanceof

Boilerplate code

- is code that tends to be duplicated throughout a section of code over and over again in a similar manner

void compareIntegers(Number number) {  
 if(number instanceof Integer) {  
 Integer data = (Integer)number;  
 System.*out*.print(data.compareTo(5));  
 }  
}

shorter syntax

void compareIntegers(Number number) {  
 if(number instanceof Integer data) {  
 System.*out*.print(data.compareTo(5));  
 }  
}

-variable data in this example is referred to as the pattern variable

- code also avoids any potential ClassCastException because the cast operation is executed only if the implicit instanceof operator returns true.

Reassigning Pattern Variables

if(number instanceof Integer data) { data = 10; }

it is a bad practice to reassign a pattern variable since doing so can lead to ambiguity about what is and is not in scope.

void compareIntegers(Number numb) {  
 if(numb instanceof final Integer data) {  
 System.*out*.print(data.compareTo(5));  
 }  
}

**number** is a target operand

**data** is the type of pattern

***Integer*** *is the name of the pattern*

***data*** *is the binding type*

**Pattern Variables and Expressions**

void printIntegersGreaterThan5(Number number) {  
 if(number instanceof Integer data && data.compareTo(5)>0)  
 System.*out*.print(data);  
}

Object obj=new String("kudzie");  
//Object obj=Integer.valueOf(200);  
if(obj instanceof String s && s.length()>0){//allows to check for the type of variable & create the right type  
 System.*out*.println(s);//kudzie  
}

void process(Object obj){  
 if(!(obj instanceof String s))  
 return;  
 System.*out*.println(s);  
 //do you code here  
}

double d=  
 switch(Shape s){  
 case Circle circle ->  
 Math.*PI*\*circle.radius\*circle.radius;  
 case Square square ->  
 square.side\*square.side;  
 default -> 0d;  
 }

double d2=  
 switch(Shape shape){  
 case null -> 0d;  
 case Circle circle when circle.radius>0 ->  
 Math.*PI*\*circle.radius\*circle.radius;  
 case Square square ->  
 square.side\*square.side;  
 default -> 0d;  
}

**Subtypes**

-type of the pattern variable must be a subtype of the variable on the left side of the expression

-cannot be the same type

Integer value = 123;  
if(value instanceof Integer) {  
 System.*out*.println((Integer) value);  
}  
if(value instanceof Integer data) {  
 System.*out*.println(data);  
} // DOES NOT COMPILE

**Flow scoping**

-means the variable is only in scope when the compiler can definitively determine its type

**-**it is not strictly hierarchical like instance, class, or local scoping

- determined by the compiler based on the branching and flow of the program

**What is the record**

-the compiler will gives a final class with setters(), getters() & toString()

-records cannot be extends coz the final

-comes with constructors & hashcode

-we cannot add ststic & instance variables

-the default constructor is called cannnonical constructor

-you can implements an interface i.e comparator

public class playWithRecord{

record Range(int begin, int end){}//begin & end is the components of the records

public static void main(String... a){

Range range= new Range(0,5);

sout("range="+ range);

sout("begin="+range.begin())

}

}

public class playWithRecord{

record Range(int begin, int end, List<Integer> values){

if(end < begin){

throw new IllegalSateException("End should be greater than begin");

}

this.begin=begin;

this.end=end;

this.values=List.copyOf(values);//this will create a mutable copy of list

Range(int end){

this(0, end);//every time you want a constructor you should call a default constructor i.e connaical constructor

}

}

public static void main(String... a){

}

}

Implementing an interface on a record

public class playWithRecord{

record Range(int begin, int end) implements Iterable<Integer>{

if(end < begin){

throw new IllegalSateException("End should be greater than begin");

}

@Override

public iterator<Integer> Iterator(){

return new Iterator<>(){

int index = begin;

@Override

public boolean hasNext(){

return index < end;

}

@Override

public Integer next(){

return index++;

}

}}

}

public static void main(String... a){

for(int value : range){

sout("value = "+value);

}

}

}

**Repetition, while and comparisons operators**

comparisons operators

- they're boolean operators

-return boolean value

<> <= >= != ==

boolean b1=10<=10; //true

boolean b2 =10!=20;//true

boolean b3= 'a'<'A';//false

'a' is greater in terms of their Unicode value

-you can compare chars by their Unicode value

boolean b3=true==10; //true

boolean b4 =false==true;//false

=whenever we use new a new object is created in the memory

== and !== will compare the references or don’t best value of objects

Cat c1=new Cat("Tom",10);

Cat c2=new Cat("Tom",10);

Cat c3=new Cat("Kudzie",20);

boolean b1= c1==c2;

System.out.println(b1);//false

boolean b2= c1.equals(c2);

System.out.println(b2);//false

c1=c2;

boolean b3= c1==c2;

System.out.println(b3);//true

boolean b4= c1==c3;

System.out.println(b4);//false

class Cat{

String name;

int age;

public Cat(String name, int age){

this.name=name;

this.age=age;

}

}

-if you want to compare objects with values youhave to override the equals() method

While

var i=1;

while(i<=10){

sout(i);//prints fro 1 to 10

i++;

}

-while loop will execute untill the condition is false i.e i is bigger than 10

-curly brackets on a while loop are optional if you have one statement

var i=0;

while(i<5) i++;//loops 4times

sout(i);//5

int i=10;

while(i<5)

sout("A");

sout("B");//not part of the while loop

Output #

B

Infinite loop

-a loop that doesnot ends

var a =10;

while(a<20){

sout("print forever");

}

while(true){

sout("execute forever");

}

while(true){ // compile error you cannot false value

sout("B");

}

-you can not have while(false) but you can have while(true)

Break & continue

continue;

-interrups the current iteration/looping

break;

-specifies the end of excuting in a switch statement and forces compiler to go out of the switch ststement or while loop

var i=1;

while(true){

sout(i);//prints 1 to 10

i=i+1;

if(i==11) break;

}

similar

var i=1;

while(i<=10){

sout(i);//print 1 to 10

i=i+1; //i++ //i+=1

}

var i=1;

while(i<=10){

i++;

System.out.println(i);//print 2 to 10

}

Implemanting continue

int i=0;

while(i<10){

i=i+1;

if(i%2==0){//2,4,6 (only even numbers)

continue;//this will cauyse the compiler to exip even number

}

sout(i);//prints 1,3 ,5,7,9

}

NB

-void the use of break and continue in the real applications in the repative loop i.e while loop

o while and logic operators

Labeling the while loop

int i=1, j=1;

A: while(i<=3){//this loop is now named A name can be lowercase and uppercase with i or more letters

B: while(j<=3){

if(j%2==0) break A;//this will force us to break from loop A i.e break out of loop A you cannot comeback

j=j+1

}

i=i+1;

}

- Avoid break and continue for when you're creating real applications

continue

continue A; tells the compiler to goback to A

int i=1, j=1;

A: while(i<=3){

B: while(j<=3){

if(j%2==0) continue A;//this will continue for ever

j=j+1

}

i=i+1;

}

while

-is called pre condition loop

do-While

-also called the POST condition loop

int i=10;

do{

sout("do");//print once coz the condition on the while is false

}while(i<10);

When the condition is true

int i=5;

do{

System.out.println("do");//this will loop for ever

}while(i<10);

7

\

while loop

-excute the condition first then execute instruction if the condtion is true

do-while

execute the instruction of the do block

**Encapsulation & Inheritance**

Encapsulation

-is the way of writing class that enables us not work with class data directly but to work with class data indirectly by using methods of data i.e getters & setters

-the accessing of class data (variable) should be only done by calling class behavior methods i.e getter & setter methods

Cat.java

public class Cat{  
 String name;  
 int age;  
}

Main.java

public class Main{  
 psvm(String... a){  
 Cat c=new Cat();  
 c1.name="Kundai";  
 c1.age=17;  
 sout(c1.name);//not allowed but compiles  
 }  
}

-the above code does not implement encapsulation coz we rare working with Class data directly

**Implementing Inheritance**

Cat.java

public class Cat {  
 private String name;  
 private int age;  
 public Cat() {}  
 public Cat(String name, int age) {this.name = name;this.age = age;}  
 public String getName() {return name;}  
 public void setName(String name) {this.name = name;}  
 public int getAge() {return age;}  
 public void setAge(int age) {this.age = age;}  
  
 @Override  
 public String toString() {return "Cat{" + "name='" + name + '\'' + ", age=" + age + '}';}  
}

Main.java

public class Main {  
 public static void main(String[] args) {  
 Cat c1=new Cat();  
 c1.setName("Kundai");  
 c1.setAge(17);  
 System.*out*.println(c1.getName());  
 }  
}

Advantages of encapsulation

-easier to check where variable is being set

-you can intercept the getter & setters’ methods

Disadvantages of encapsulation

-you cannot decrement/increment the variable using getter & setter methods

NB

-good practice to adhere with Naming standards of getter & setter methods coz some framework in java will be easy to identify the methods

**Inheritance**

Main.java

public class Main {

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

A a1= new A();

a1.x=10;

a1.m();

B a2= new B();

a2.x=20;

a2.m();

A a3= new B();

a3.x=30;

a3.m();

}

}

//A.java

class A{

//class members are inherited depending with access modifiers and if they are not final

//default varibles are only inherited by classes in same package

int x;

private int y;//note visible to subclass

double area;

void m(){

System.out.println("method m"+x);

}

}

//B.java

class B extends A{

String area;//this will hide the inherited area variable

public B(){

sout(super.area);

}

}

Output:

method m10

method m20

method m30

What you can do after overriding

-add new fields

-add new behaviour

-change fields-> hiding fields

-change behaviour -> overriding

-inheritence is also called is-A i.e the relationship btwn two objects

e.g class Circle extends Shape{}

Has-A relationship

-is when another objects contains another objects

public classs Owner{

private Laptop lap;

}

Overriding

-changing the behavour of the overriden method

-you cannot extends multiple classes in java

-public class member can be vissble to any subclass

-default can be visble to the subclass of subclass in same class

-protected can be visible from any subclass in the project

Overidding Rules

method signatures of method in super/subclass should be same i.e method name/identifier and order /type of parameters should be same

-Return type methods should be same and in case of object return types they can be same or an overidden methods can have subtype of the return type

method in superclass

Number m2(){//supertype return type

return 10;

}

method in subclass

Integer m2(){//subtype return type

return 5;

}

Order of access modifiers

public protected

default

private

-the overidden method should have an access modifer which more visible than the one in the super class methods or same

SuperClass

A.java

public class A{

protected void m3(){

sout("in A");

}

}

SubClass

B.java

public class B extends A{

public void m3(){

sout("in A");

}

protected void m3(){

sout("in A");

}

void m3(){//will not compile

sout("in A");

}

private void m3(){//will not compile

sout("in A");

}

}

public class Main {

public static void main(String[] args) {

A a1= new A();

a1.m();

A a3= new B();

a3.m();

}

}

//A.java

class A{

void m(){

System.out.println("in class A");

}

Number m2(){

return 10;

}

}

//B.java

class B extends A{

/\*void m(){

System.out.println("in class B");

}\*/

@Override

Integer m2(){//integer is te child of Number parent

return 5;

}

}

Output:

in class A

in class B

When there is no overriden method in sub class the subclass object will refrence the method in super class

public class Main {

public static void main(String[] args) {

A a1= new A();

a1.m(); //in class A

A a3= new B();

a3.m();////in class A

}

}

//A.java

class A{

void m(){

System.out.println("in class A");

}

}

//B.java

class B extends A{

}

Output

in class A

in class A

Overidding methods that throws the exceptions

Throwing checked exception

Superclass

A.java

public class A{

void m() throws IOException{

}

}

SubClass

B.java

public class B extends A{

void m(){

//avoid throwing exception from the superclass method

}

void m()throws FileNotFoundException{

//you can throw subtype exception

}

void m()throws IOException{

//you can throw same type exception

}

void m()throws Exception{//will not compile

//can't throws wider/bigger/type of exception

}

}

-@Override annotation is used to explicity show that the method has being overriden

-in the ocp11 exam @Override can be shown or note

comparing overloading & overriding

overloading

-having multiple methods with same (identifier) names with different paratemers -parameters can be in different order

-overloading methods are sually declared in same class

-constructors can be overloaded

-static or non static my be overload

-doesn't care about access modifier

-doesn't care about about exception

--doesn't care return types

-you can overload final method

-cannot overload with inhertence

overriding

-same method identifer

-takes care of the return type

-no overriding for constructor

-takes care of exception

-use of correct access modifer to override

-final method can not be overriden

-final class can not be extended

-you cannot override static method

-override is achieved through inheritence

Array

-stores list of constants

-store values of same type

-collection of fixed values of the same type

declaring an array

int [] x = new int[3];

Position of square bracket

int x1[];

int [] x2;

multiple declaration

int [] z1,z2;// both will inherit what is after the type (int) i.e []

z1=new int[4];

z2=new int[5];

Brackets after in the multiple declaration

int w1[], w2;

//wi is an array

//w2 is a primitives

w1=new int[5];//int[]

w2=10;//int

int [] r [];//same int [][] r;// int r [][];

int [] a1[], a2;

a1=new int[3][4];//a1 inherited two []

a2= new int[2];//a2 inherited one []

Adding elements into the array

x[0]=2;

x[1]=3;

x[2]=4;

Loop through the array

for(int i=0;i<x.length;i++){

sout(x[i]);

}

Intializing the array

int[] y={};//empty array with alength 5

int [] y2={2,8,7,9,4};

NB

-this type of declaration will only be used in the intialization

int [] z;

z={}//this will not compile

Solution

int [] w1;

w1= new int[] {2,8,7};//this will compiles but you don't haveto specify the size

int [] w2;

w2= new int[3] {2,8,7};//will not compile

Declaring using var

var q1= new int[10];

var completely replace the type of the variable and the square brakets

you can not use the curly brackets with var

var q2={1,2,3}; //compile error

var q3 = new int[] {1,2,3,4};//compiles create an array length of 4

Setting an array as a method parameter

public void m(int [] z){

}

var z=new int[] {77,84,32,109};

m(z);

directly passing method parameter

m(new int[5]);

m(new int [] {54,674,43,12})

M({1,2,3,4});//will not compiles

-you can not var as a method parameter and method return type

public var m1(var x){}//will not compiles

ArrayIndexOutOfBoundsException

-thrown when you try to access an array value using an index which is out of range

int [] b= {4,65,7,9};

sout(b[10]);//throws ArrayIndexOutOfBoundsException

Enhenced for loop

for(type of the value:collection){

}

var x= new int[]{1,2,3,4,5};

for(int a : x){//type : array //a is varaible value for the element values in the array

a=10;//you can not change the values in the array so the array remains the same

sout(a); //10 10 10 10

}

-you can manage the control of a loop using break

-enhanced forloop avoids the risks of infinite loops

-can not used to change values in the array

var x= new int[]{23,40,78,56};

int i=0;

for(int a:x){

a=100;//this will only change the temporary value of a in the for loop only

System.out.println("number"+a);// 100 100 100 100

}

for(int a:x){

i++;

x[i-1]=a\*2;//this will change the elements in the array i.e array values

System.out.println("number"+a); //200 200 200

}

for(int a:x){

System.out.println("number"+a);//46 80 156 112

}

for(final int a:x){

//a=100;//will not compiles coz a is a constant you cannot reassign the value

System.out.println("number"+a);

}

final int [] x= {1,2,3,4};//means that x is a constant but values in the array are not constant

[2]=23;//this posisble to reassign

x=new int[] {7,8,9,8};//this will not compile coz you cannot assign constant

String [] y={"AA","CB","D","VV"};

- you can have an array of any objects

Multiple dimension array (Matrix)

for ocpjp11 you will be asked mainly two dimention array but its good to know 2d+ array

int [] [] z= {4,5,3};//will not compile you should have arrays inside an array

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

sout(z.length);//3

sout([0].length);//4

int [][][] a ={{ {1,2}, {} },{{1}},{null},null};

sout(a.length);//4

sout(a[0].length); //2

sout(a[1].length); //1

sout(a[2].length); //1 coz null is a value that represent nothing value

sout(a[3].length); //throws NullPointerException coz you are trying to access nothing

Enhenced for loop for 2D array

int [][] a={ {14},{13,12},{1} };

int [][] a= new int[][]{ {14},{13,12},{1} };

for(int [] x: a){

sout(x.length);//1 2 1

for(int y:x){

sout(y+ " ");//14 //13,12 //1

}

}

More decalaration of 2D array

int [][] a={ {14},{13,12},{1} };

int [][] a= new int[][]{ {14},{13,12},{1} };

int [][] x= new int[number\_of\_array\_in\_main\_array][number\_ofvalue\_in\_each\_array];

int [][] x= new int[3][];//[null,null,null]

x[0]=new int[]{1};

x[1]=new int[]{2,3,4};

x[2]=new int[]{10};

//{ {1}, {2,3,4}, {10} }

for(int [] i:x){

for(int j:i){

sout(j);

}

}

int [][] y= new int[3][2];//{ {0,0},{0,0},{0,0} } it will store default values for int its 0

int [][] a= new int[3][2];

a[0]=new int[]{1};

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

a[2]=new int[]{10};

for(int [] c:a){

System.out.println(c.length);

for(int z:c) System.out.println("inner value"+z);

}

Output

1

inner value1

3

inner value2

inner value3

inner value4

1

inner value10

m2(new int[]{1,2,3,4,5});

void m2(int[] a){

for(int x:a){

System.out.println(x);//print 1 2 3 4 5

}

}

Varargs

-varriables number of argurments or parameters

-3 dots represents the square brackets []

dclaration of varargs

type(three dots) identifier

int... a //this can be treated as an array

2d varargs array declaration

int[]... x

void m1(int... a){

If(a.length>0){

sout(a[0]);//1 10

}

for(int x:a){

sout(x);

}

f

}

m1();//throws ArrayINDEXOUTOfBoundException

m1(1);//the parameter should be of same type

m1(1,2,3,4,5,6,6,7,8,9);

NB

- you can only have one varargs parameters in a methods if there are many different parameters varargs should be always the last

void m2(String y , int... x){

sout(y);

sout(x.length);

}

m2("ABC");//ABC 0

m2("ABCD",2,3,4);//ABCD 3

void m2(int... x,String y){}//compile error varargs should the last

m2(new int[]{1,2,3,4,5});

m2(1,2,3,4,5);//will not compile

void m2(int[] a){

for(int x:a){

System.out.println(x);//print 1 2 3 4 5

}

}

m2(new int[]{1,2,3,4,5});

static void m2(int... a){

for(int x:a){

System.out.println(x);

}

}

2D varargs

void m3(int[]... x){//array of arrays->matrix

for(int[] a:x){

sout(a);

}

}

int [] b1={9,8,7,6};

int [] b2={1,2,3,4};

int [][] c={b1,b2}

m3(c);

var a1=new int[]{1,2,3,4,5};

var a2=new int[]{11,22,33,44,55};

int[][]c={a1,a2};

m2(c);

static void m2(int[]... a){

for(int[] j:a){

for(int z:j){

System.out.println(z);

}

}

Parameter of a java main method

public static void main(String [] args){}

public static void main(String args[]){}

public static void main(String... args){}

-if any thing is changed from the above synatx of main method above this will be invalid main()

-parameters names can have different names

ping google.com

-ping is the name of the program that can accept a String

-String [] args parameter is never a null and in most cases array length is zero

public static void main(String [] args){

sout(args.length);//0

for(int a:x){

sout(a);

}

}

Accessing program arguments in intelliJ

>Edit configuration

static method declaration

-static methods belongs to the class not to the object

-static variable or methods not owned by the instance but owned by the class

calling with instance

Main.java

psvm(String... args){

var cat=new Cat();

cat.sayHello();

or

Cat c1=null;

c1.sayHello();//this will result on the NullPointerExeption coz c1 object is null

}

Cat.java

public class Cat{

String name;

void sayMeow(){

sout("Meow ! My name "+this.name);//this referes to the current instances that is new Cat()

}

}

Calling using class

Main.java

psvm(String... args){

Cat.sayHello();

or

Cat c1=null;

c1.sayHello();//this will compile cz the method is part of the class which is the type

}

Cat.java

public class Cat{

String name;

static void sayMeow(){

sout("Meow ! My name "+this.name);//compile error sayMeow is now part of the class not part of the instance

}

}

**STRING AND STRINGBUILDER**

String

-a sequence of characters

*String name1= "Kudzie";*

*String name2 = new String("Kundai"); // “Kundai” is the String laterals*

- it implements Serializable, CharSequence and Comparable interface

-the length and capacity of the String is fixed

-length of String is same with the capacity

-String is the most important class in java

-the most commonly used objects is String in java projects

-most compulsory questions in java comes from String

-many questions in OCP exam

**String Interning**

-**storing only one copy of each literal String in the pool**

**-**optimize the amount of memory allocated

-S*tring* variable and assign a value to it, the JVM searches the pool for a String of equal value.

-i**f found, the Java compiler will simply return a reference to its memory address, without allocating additional memory.**

String constantString1 = "Baeldung";  
String constantString2 = "Baeldung";  
System.*out*.println(constantString1==constantString1); //true  
System.*out*.println(constantString1.equals(constantString2));//true

**Strings Allocated Using the Constructor**

-When we create a String via the new operator, the Java compiler will create a new object and store it in the heap space reserved for the JVM.

String constantString = "Baeldung";  
String newString = new String("Baeldung");  
System.*out*.println(constantString==newString);//false "==" is for reference/address comparison  
System.*out*.println(constantString.equals(newString));//true String equals ifs for content comparison

## String**Literal vs**String Object

-**create a String object using the new() operator, it always creates a new object in heap memory and** put the object in the string pool for future re-use

**-if we create an object using String literal syntax e.g. “Baeldung”, it may return an existing object from the String pool, if it already exists**

String first = "Baeldung";  
String second = "Baeldung";  
System.*out*.println(first == second); // true coz the first & second variable they're pointing to same object (references/address) in String Pool

String third = new String("Baeldung");  
String fourth = new String("Baeldung");  
System.*out*.println(third == fourth); // false coz they point to diffrence refrences in the HEAP memory

**we should use the String literal notation when possible**

**-**easier to read and it gives the compiler a chance to optimize our code.

## **String Manual Interning** -String will store its reference in the pool and the JVM will return this reference when needed -done by calling the intern() method on the object we want to intern

String constantString = "interned Baeldung";  
String newString = new String("interned Baeldung");  
System.*out*.println(constantString==newString);//false  
System.*out*.println(constantString.equals(newString));//true  
  
String internedString = newString.intern();));//store String into StringPool  
System.*out*.println(constantString==internedString);//true  
System.*out*.println(constantString.equals(internedString));//true

## **Garbage Collection** Before Java 7,-JVM **placed the Java String Pool in the PermGen space, which has a fixed size — it can’t be expanded at runtime and is not eligible for garbage collection** -risk of interning Strings in the PermGen (instead of the Heap) is that **we can get an OutOfMemory error** from the JVM if we intern too many Strings.

From Java 7 onwards,

-Java String Pool is **stored in the Heap space, which is garbage collected** by the JVM

-**reduced risk of OutOfMemory error** because unreferenced Strings will be removed from the pool, thereby releasing memory.

Interview questions

1. Why String Pool concept is available to String Object not to StringBuffer ?

if you're regular customer (String) to a Bar you get special treatment

-String objects are the most used object in java

2. Why string objects are immutable where as StringBuffer objects are mutable ?

String are immutable coz of the String Pool that allows object to be reused so objects should not be changed

3 In addition to String Objects any other objects are immutable in java ?

All wrapper objects are immutable

Important constructors of String class

Creating an empty String object

String s= new String();  
System.*out*.println(s.length());//0  
  
String sNull= null;  
System.*out*.println(sNull.length());//NullPointerException

Creating a String object using laterals

String s2= new String("kudzie");

An object will be created in the HEAP memory

Second object will be created in String pool if the "kudzie" does not exist

Creating a string from StringBuffer

//String s= new String(StringBuffer sb);  
String ssbf= new String(new StringBuffer());

Creating a String objects from StringBuilder

//String s= new String(StringBuilder sb);  
String ssbd= new String(new StringBuilder());

Creating a String objects from Characters array

//String s= new String(char [] ch);  
String sChar=new String(new char[]{'j','a','v','a'});  
char [] ch={'k','u','n','d','a','i'};  
String sChar2=new String(ch);  
System.*out*.println(sChar); //java  
System.*out*.println(sChar2);//kundai

Creating a String Objects from byte[]

//String s= new String(byte [] a);  
byte [] sByte={100,80,-23,56,67,99};//range of byte is -128 to 127 if the number is out of the range you need to cast it first to avoid error  
String bStr= new String(sByte);  
System.*out*.println(bStr);//dP�8Cc

byte [] sByte={100,80,-23,56,67,99,(byte) 300};//range of byte is -128 to 127  
String bStr= new String(sByte);  
System.*out*.println(bStr);//dP�8Cc,  
System.*out*.println(new String(new byte[]{-127}));  
System.*out*.println((byte) 300);//44 why ?

**Difference btwn String & StringBuffer vs StringBuilder**

String

-string objects are immutable means objects cannot be changed after creation

e.g

String s= new String("kudzie");

s.concat("software");

sout(s); // kudzie i.e once we create String objects we are not allowed to perform changes but if we try to change the new object will created but will not change the Old String objects

StringBuffer

-StringBuffer objects are mutable means objects can be changed after creation

StringBuffer sb= new StringBuffer("kudzie");

sb.append("software");

sout(sb); // kudzie software //we can perform any changes no new object will created

Difference between String and StringBuffer == Operator Vs equals method

String

String s1=new String("kudzie");

String s2=new String("kudzie");

sout(s1==s2); //false coz the reference are pointing different objects

sout(s1.equals(s2)); //true coz equals() in String class is for content comparison

StringBuffer sb1= new StringBuffer("kudzie");

StringBuffer sb2= new StringBuffer("kudzie");

sout(sb1==sb2); //false

sout(sb1.equals(sb2)); //false coz equals() in StringBuffer class is for refence/address comparison

Double equal sign "=="

-its for refrenece/address comparison

-also called refrence operator

-compare the reference of the object

equals() method in object class is meant for refrence/address comparison

But equals() method in String class is overriden from Object class to do content comparison

equals() method in StringBuffer will call equals() method in Object class which is for refrence/address comparison

String Objects Creation Heap and String Constant Pool (SCP)

- the default pool size from Java 7u40 until Java 11 was 60013 and now it increased to 65536

-for any string laterals is stored in the String pool

String s0= new String String("abc");

Object s0 is stored in HEAP memory and another copy in String Pool coz of String laterals

difference btwn String s= new String String("kudzie"); and String s="kudzie";

String s= new String String("kudzie");

-a new String object will be created in the HEAP Memory coz of the keyword

-another copy of String object will be create into String pool for future use coz the String object has a literal as parameter

String s="kudzie";

-a new string will be created in the String pool

-String pool is now moved to HEAP memory that can be extended

-Object in the String Pool is not valid for garbage collection

-if the object is already in the String Pool the new object will reference the old object

-But if the object is not there in the String Pool a new object will be created

-every runtime operations creates object only in the HEAP memory not in the String Pool

-String leterals in the String methods parameters are created in the String Pool

String s1= new String("kudzie");//1 object will created in the HEAP memory coz of "new" keyword

String s2= new String("kudzie");

String s3="kudzie";

String s4="kudzie";

|  |  |
| --- | --- |
| HEAP MEMORY | STRING POOL |
| S2  kudzie  S1  kudzie | For future use  S4  S3  kudzie |

How many objects in the String Pool

> 1 object

How many objects in the HEAP memory

> 2 objects

Example

String s= new String("kudzie");

s.concat("madziva");

s=s.concat("solutions");

|  |  |
| --- | --- |
| HEAP MEMORY | STRING POOL |
| 🗶 objects are valid for garbage collection  🗶  🗶  kudziesolutions  kudziemadziva  s.concat("madziva");  s  kudzie | Coz of String lateral  Coz of String lateral  software  Coz of String lateral  madziva  kudzie |

String s1= new String("Spring");  
s1.concat("Fall");  
String s2= s1.concat("Winter");  
s2.concat("Summer");  
  
System.*out*.println(s1); //Spring  
System.*out*.println(s2); //SpringWinter

|  |  |
| --- | --- |
| HEAP MEMORY | STRING POOL |
| 🗶  S1.concat("Fall");  S2.concat(“Summer”);  SpringWinterSummer  🗶 objects are valid for garbage collection  S2  SpringWinter  Fall  🗶  S1  Spring | Coz of String laterals  Summer  Summer  Coz of String lateral  Winter  Coz of String lateral  Fall  Spring  Coz of String lateral |

public static void main(String[] args) {  
 System.*gc*();  
}  
  
@Override  
protected void finalize() throws Throwable {  
 System.*out*.println("garbage collection");  
}

String s10= new String("You cannot change Me");  
String s11 = new String("You cannot change Me");  
System.*out*.println(s10==s11); //false coz s10 &s11 are pointing to different refrence/address  
  
String s13="You cannot change Me";  
System.*out*.println(s11==s13); //false coz s11 & s13 they are pointing different address/references  
  
String s14= "You cannot change Me";  
System.*out*.println(s13==s14); //true  
  
String s15="You cannot "+"change Me"; // "You cannot ".concat("change Me"); no object will be created but s15 will point to Object already in String Pool

//the operation to get s15 will be performed at compile time not at runtime  
System.*out*.println(s14==s15); //true  
  
String s16="You cannot ";  
String s17=s16+"change Me"; //s16.concat("change Me"); s17 operation will be performed at runtime coz there a variable s16  
System.*out*.println(s14==s17); //false coz s14 & s17 is pointing to different address/reference  
  
final String s18="You cannot ";  
String s19 = s18+"change Me"; //s18.concat("change Me"); the operation to get s19 will be performed at compile time coz s18 (final variable i.e constant) & “change Me” are both constants  
System.*out*.println(s14==s19);//true coz s14 & s19 is stored in the SCP and they point to same address/reference

|  |  |
| --- | --- |
| HEAP MEMORY | STRING POOL |
| S19  S18  You cannot change Me  S17  S16  S13  S14  S15  You cannot change Me  S11  You cannot change Me  S10  🗶  🗶 | change Me  You cannot  You cannot change Me  Coz of String laterals  Summer  Summer  Coz of String lateral  Coz of String lateral  Coz of String lateral |

String sc1= new String("kudzie");  
String sc2=sc1.toLowerCase();//this will cause an object to created in the HEAP memory  
String sc3=sc1.toUpperCase();  
System.*out*.println(sc1==sc2);//true coz sc1 & sc1 points to "kudzie" in StringPool  
System.*out*.println(sc1==sc3);//false  
  
System.*out*.println(sc1.equals(sc2));//true coz equals is content comparisons in String class  
System.*out*.println(sc1.equals(sc3));//false

|  |  |
| --- | --- |
| HEAP MEMORY | STRING POOL |
| KUDZIE  Sc3  Sc2  sc1  kudzie | Coz of String lateral  kudzie |

Advantages/Disadvanatages of String Constant Pool (SCP)

Advanatages of String Pool

-allows to objects to be reused by different references/address in the String Pool which results in improved performance by using less memory due to creation of few objects

- saves time to create a new string if there is already a string with the same value present in the pool

- Cache improves performance and reduces memory usage

Disadvantages of String Pool

-increases the complexity coz the objects is being used by many references

**CONCATENATION**

-merge/ add two or more things together

**RULES**

1. if both operands are numeric + means addition

2. if either operand is String + means concatenation

3. The expression is evaluated from left to right

- immutability - unchangeable

-one the string object is created the string cannot be changed

-String class is final to prevent other classes from extending and enhance immutability(not changing).

-method changing - return class object

**jshel** is the best no need to create a class

*cdm*

*cd to a directory*

*>jshell*

*save*

*/save nameofthefile.jshell*

*/save nameofthefile.jsh*

*/open nameofthefile.jsh*

*/exit*

*/list*

System.*out*.println("a"+"b"); //ab  
System.*out*.println("a"+"b"+3); //ab3  
System.*out*.println(1+3); //4  
System.*out*.println(1+2+"c"); //"3c"  
String s2="1";  
s2.concat("3"); //12  
System.*out*.println(s2); //1

**IMPORTANT METHODS** - String

**public boolean isEmpty()**

-check whether the String Object does it have characters or not but the empty space is regarded as characters

String s0= "";  
String s1= " ";  
String s2= new String();//even if you leave space the s2 will empty  
String s3= new String( );  
System.*out*.println(s0.isEmpty());//true  
System.*out*.println(s1.isEmpty());//false space is regarded as char  
System.*out*.println(s2.isEmpty());//true  
System.*out*.println(s3.isEmpty());//true  
  
System.*out*.println(s0.isBlank());//true  
System.*out*.println(s1.isBlank());//true  
System.*out*.println(s2.isBlank());//true  
System.*out*.println(s3.isBlank());//true

**public** **int length()**   
- determine number of chars in String

String s0= "";  
String s1= " ";  
String s2= new String();//even if you leave space the s2 will empty  
String s3= new String( );

System.*out*.println(s0.length());//0  
System.*out*.println(s1.length());//4  
System.*out*.println(s2.length());//0  
System.*out*.println(s3.length());//0

**Note**

String used String.length() but arrays use array.length

int [] intArray={1,3,778,8,0};  
System.*out*.println(intArray.length);//5

String s0= "";

System.*out*.println(s0.length());//0

**char charAt(int index)**   
- char at the specific index

**int indexOf(char ch)**  
 -return first index of desire value

String str="kudzie Madziva";  
System.*out*.println(str.indexOf('z'));//3 //space is counteted  
System.*out*.println(str.indexOf("a"));//8 //if the character appears more than once it will return the indeOf the first letter

System.*out*.println(str.indexOf("g"));// returns -1 coz there is no character ‘g’

**int lastIndexOf(char ch)**  
 -return the last index of desire value

String str2="kudzie Madziva";  
System.*out*.println(str2.lastIndexOf('z'));//10  
System.*out*.println(str2.lastIndexOf("a"));//13  
System.*out*.println(str2.lastIndexOf("g"));//-1

**public String toLowercase()**

-return String in lowercase i.e small letters

String str3="KUNDAi";  
System.*out*.println(str3.toLowerCase());//kundai

**public String toUppercase()**

-return String in uppercase i.e capital letters

String str4="kudZanai";  
System.*out*.println(str4.toUpperCase());//KUDZANAI

**int indexOf(int ch,int fromIndex)**

**int indexOf(String str)**

**int indexOf(String str, int fromindex)**

**public String trim()**

-removes blank space at the beginning and end of the String then return String

-trim() does not removes blank spaces between the Strings

String str5=" Kudzai Madziva ";  
System.*out*.println("length before trim() "+str5.length());//length before trim() 31  
System.*out*.println(str5.trim());//Kudzai Madziva  
System.*out*.println("length after trim() "+str5.trim().length());//length after trim() 14

Use case of trim() & toLowercase()

Scanner sc= new Scanner(System.*in*);  
System.*out*.println("Enter your birth city name");  
String inStr= sc.nextLine().toLowerCase().trim();  
System.*out*.println(inStr=="chinhoyi");//false even if the inStr="chinhoyi" coz it refrencing from different String objects  
if(inStr.equals("chinhoyi")){  
 System.*out*.println("Your was born from "+inStr);  
}else if(inStr.equals("harare")){  
 System.*out*.println("Your was born from "+inStr);  
}else if(inStr.equals("mutare")){  
 System.*out*.println("Your was born from "+inStr);  
}else{  
 System.*out*.println("Invalid city name "+inStr);  
}

**public String substring(int beginIndex)** i.e “substring”in lowercase  
 - returns part of string from beginIndex(inclusive) to the end of the String

String ss0= " abcdefg";  
String ss1= "abcdefg";  
System.*out*.println(ss0.substring(2));//bcdefg JVM also count the space  
System.*out*.println(ss1.substring(2));//cdefg

**public String substring(int beginIndex,int endIndex)**

inclusive exclusive

-returns part of the String from beginIndex(inclusive) to the endIndex-1 (exclusive)

System.*out*.println(ss0.substring(3,6));//cde 3to5  
//System.out.println(ss1.substring(3,600));//throws StringIndexOutOfBoundsException coz 600 is not part of the range  
System.*out*.println(ss1.substring(3,6));//def 3to5

String toString()

String str="animal";

to see all string methods

>jshell> str.

str.length();//7

str.charAt(0);//a

str.indexOf("a");//0

str.indexOf("al");//4

str.indexOf("al",4);//-1 coz from index 4 there is no al

str.substring(3);//mals its start from 3 to the end

str.substring(3,4);//m

str.substring(3,3);//this rturns empty

String methods

String toLowerCase() and String toUpperCase()

boolean equals(Object obj) and boolean equalsIgnoreCase(String str)

boolean startsWith(String prefix) and boolean endsWith(String suffix)

**public String replace(CharSequence old, CharSequesnce new)**

-used to replace old chars with new characters

String baba= "baba";  
System.*out*.println(baba.replace('b','m'));//mama mixing double quote not allowed single quote for one character double quaote for everything  
System.*out*.println(baba.replace("b","m"));//mama coz there is no matching character to replace it will return the original String  
String kundai= "kundai";  
//System.out.println(kundai.replace('nd','dz'));//will not compile  
System.*out*.println(kundai.replace("nd","dz"));//kudzai

boolean contains(CharSequence cs)

String trim() and String strip() - remove whitespaces from & end

String stripLeading() - remove whitespaces from beginning

String stripTrailing() - remove whitespaces from end

String interm() - returns the value from string pool if it is there otherwise it add the value to the string pool

str.toUpperCase();

str.toLowerCase();

"ABC".equals("ABC");//true

"abc".equalsIgnoreCase("ABC");//true

"abcde".replace('a','A');//Abcde

"abc".startsWith("a");//true

"abc".startsWith("A");//false

"abc".endsWith("c");//true

"abc".contains("b");//true

"abc".strip();//this removes space at the begining and end but not in btwn

strip() supports unique code

Creation of Our Own Immutable Class

-An object that we cannot be change the content once te object is created

-if there is changes in the objet's content a new object will be created

-But if there is no changes in the Object content a new object will be created

public final class ImmutableObject {//Immutable class are final coz noone is allowed to change the the class  
 private int i;  
 ImmutableObject(int i){  
 this.i=i;  
 }  
 public ImmutableObject modify(int i){  
 if(this.i==i){  
 return this;  
 }else{  
 return new ImmutableObject(i);  
 }  
 }  
}

public class MainImmutable {  
 public static void main(String[] args) {  
 ImmutableObject a1= new ImmutableObject(10);  
 ImmutableObject a2= a1.modify(100);  
 ImmutableObject a3 = a1.modify(10);  
 System.*out*.println(a1==a2);//false coz they refrence/address is not same  
 System.*out*.println(a1==a3);//true coz the refrence/address are same  
 }  
}

final Vs immutability

final refers to refrence variable

-if you don't want to reassign a variable use FINAL

immutable refres to objects

-if you don't want to hange the content of object use Immutable

final StringBuffer sb= new StringBuffer("kudzie ");//by making the refrence variable it does not mean the "sb" object is now immutable  
//final is prevent the reassignment of the "sb" refrence variable in java  
sb.append("java programmer");  
sb= new StringBuffer("Kundai ");//compile error coz you can not assign FINAL variable  
System.*out*.println(sb);//kudzie java programmer

Which of the following are meaningful?

1 final variable //valid

2 final object //in valid you can change to object

3 Immutable variable // no such object

4 Immutable //valid

StringBuffer

-the length is the current number of characters in the String and capacity of the String is unknown

Need of StringBuffer

-if the content is continous changing we can use StringBuffer

-if the content is fixed does not change many times you can use String

-for String for any changes to the content of object the new object will be created results in using large memory

-For StringBuffer for any changes to the content will be perfomed to the to the existing object only this saves memory

StringBuffer class Constructors

StringBuffer sb= new StringBuffer();

StringBuffer sb0= new StringBuffer();//the default capacity is 16  
System.*out*.println(sb0.capacity());//16  
//if a bigger string is added to the to sb0 with a capacity of 16 a new object willl be created by JVM then the old object is garbage collected  
//new capacity = (current capacity+1)\*2  
// new capacity=(16+1)\*2=34  
sb0.append("abcdefgefjklmnop");  
System.*out*.println(sb0.capacity());//16  
sb0.append("krswzy");  
System.*out*.println(sb0.capacity());//34=(16+1)\*2

StringBuffer sb= new StringBuffer( int intialCapacity);

//Creating StringBuffer with intial capacity which is good for perfomance  
StringBuffer sb1= new StringBuffer(5);  
System.*out*.println(sb1.capacity());//5  
sb1.append("'kudzanai 2");  
System.*out*.println(sb1.capacity());//12= (5+1)\*2

StringBuffer sb= new StringBuffer(String s);

StringBuffer sb3=new StringBuffer("kudzie");  
System.*out*.println(sb3.capacity());//capacity=sb3.length()+16=22;  
sb3.append("abcdefgefjklmnopj1234567");//46=(22+1)\*2  
System.*out*.println(sb3.capacity());

Important Methods of StringBuffer class

public int length()

-return the number of characters together with empty space

public int capacity()

public char charAt(int index)

-return the character at a given index

StringBuffer sb0= new StringBuffer("Kundai");  
System.*out*.println(sb0.charAt(3));//d  
System.*out*.println(sb0.charAt(20));//capacity=6+16=22 //if the index is more than sb0.length()-1 the program will throw StringIndexOutOfBoundsException in case of String but for StringBuffer it throws StringIndexOutOfBoundsException there is no StringIndexOutOfBoundException in java

StringBuffer sb1= new StringBuffer("Kud zanai");  
System.*out*.println(sb1.charAt(3)+"str");//returns emptyspace character

Public void setCharAt(int index, char ch)

-return the character at a given index

StringBuffer sb1= new StringBuffer("Kud zanai");  
sb1.setCharAt(3,'x');//you can only wrap characters with single quotes only to avoid compile error  
System.*out*.println(sb1);

Public StringBuffer append(Any)

-returns StringBuffer after combining object together

StringBuffer sb2= new StringBuffer("Kuku12345");  
sb2.append(PrimitiveData type value);  
sb2.append(char [] str);  
sb2.append(String s);  
sb2.append(Object obj);  
sb2.append(CharSequence charS);  
sb2.append(char [] s,int start ,int end);  
sb2.append(StringBuffer sb);

StringBuffer sb3= new StringBuffer();  
sb3.append("Is PI value is ");  
sb3.append(3.1d);  
sb3.append(4);  
sb3.append(" is the correct statement ? ");  
sb3.append(true);  
System.*out*.println(sb3);//Is PI value is 3.14 is the correct statement ? true

StringBuffer sb11=new StringBuffer("kuku");  
sb11.append(new StringBufferConstructors());//kukuStringBufferConstructors@34a245ab if the object does not implement it will print the object Hashcode  
System.*out*.println(sb11);

public StringBuffer insert(int offset,AnyDataType str)

-returns a StringBuffer after inserting the String or primitive value at a given index

StringBuffer sb4= new StringBuffer("Kudzanai");  
System.*out*.println(sb4.insert(3,"12345"));//Kud12345zanai  
System.*out*.println(sb4.insert(9,true));//Kud12345ztrueanai  
System.*out*.println(sb4.insert(15,0.678f));//Kud12345ztruean0.678ai

public void delete(int startIndex, int endIndex)

-return StringBuffer object after deleting the charcters from beginIndex to endIndex-1

StringBuffer sb5= new StringBuffer("abcdefgh");  
System.*out*.println(sb5.delete(3,6));//abcgh coz delete characters from index 3 to 6-1 index

public StringBuffer deleteCharAt(int index)

-returns a StringBuffer after removing the charcter at specified index

StringBuffer sb6= new StringBuffer("abcdefgh");  
System.*out*.println(sb6.deleteCharAt(6));//abcdefh coz the char at index will be deleted

Public StringBuffer reverse()

-returns a reversed StringBuffer object

-In String there is no reverse() mthod but in StringBuffer there is reverse()

StringBuffer sb7= new StringBuffer("abcdefgh");  
 System.*out*.println(sb7.reverse());//hgfedcba in String class there is no reverse

public void setLength(int len)

-set the length of the StringBuffer the remain characters are removed

StringBuffer sb8= new StringBuffer("Kudzai Madziva");  
sb8.setLength(6);  
System.*out*.println(sb8);//Kudzai  
System.*out*.println(sb8.length());//6

public void ensureCapacity(int ensureCapacity)

-used to set the capacity dynamically

StringBuffer sb9= new StringBuffer();  
System.*out*.println(sb9.capacity());//16 is the defult capacity is 16  
sb9.ensureCapacity(1000);  
System.*out*.println(sb9.capacity());//1000  
  
StringBuffer sb10=new StringBuffer("str");  
System.*out*.println(sb10.capacity());//3+16=19

public void trimToSize()

-this remove unnecessary space when you set the capacity of String to be big value when the content length is very smaller than

StringBuffer sb12= new StringBuffer(2000);  
sb12.append("Kundai Madziva");//this very small to be saved in the in sb12 with a capacity of 2000 why the String has only  
System.*out*.println(sb12.capacity());  
sb12.trimToSize();//remove unoccupied empty space in the StringBuffer sb12  
System.*out*.println(sb12.capacity());//14

StringBilder

-is non synchronized version of StringBuffer

- a sequence of characters

StringBuilder sb1= new StringBuilder();//create an empty string object length is 0 and the capacity is 16 coz it aadd an extra space of 16

StringBuilder sb2= new StringBuilder("animal");//length is 7 capacity is 23

StringBuilder sb3= new StringBuilder(10);

-it implements interface CharSequence

-Mutability- changeable

-Method Chaining -return class object

- converting String builder object to String

StringBuilder sb2= new StringBuilder("animal");

sb4.toString();

**Need of StringBuilder**

StringBuffer methods are the synchronized means that threads are executed one at time

-Due to non concurrent execution StringBuffer methods StringBuilder was introduced since it allows concurrent execution of methods to the StringBuilder

String builder methods

StringBuilder append(String str)

- add parameter to a StringBuilder

StringBuilder insert(int offset,String str)

- add character

StringBuilder delete(int startIndex, int endIndex)

- remove character

// inclusive exclusive

StringBuilder deleteCharAt(int index)

StringBuilder replace(int startIndex, int endIndex, String newString)

// inclusive exclusive

StringBuilder reverse()

String alpha = "";

for(char c='a';c<='z';c++){

alpha+=c;

}

same code in SpringBuilder

StringBuilder sb1= new StringBuilder()

for(char c='a';c<='z';c++){

sb1.append(c);//ther is not string concat StringBuilder

}

StringBuilder a= new StringBuilder("abc");

StringBuilder b=a.append("de");//abcde

b=b.append("f").append("g");//abcdefg

sout(a);//abcdefg

sout(b);//abcdefg

a and b are same oz they are refrencing same object

StringBuilder a= new StringBuilder("kudzie");

a.insert(0,"-'");//-kudzie

a.insert(6,"-'");//kudzie-

StringBuilder x= new StringBuilder("abcdef");

x.delete(1,3);//adef this delete upto value les than 3

x.deleteCharAt(3);//ade

StringBuilder y= new StringBuilder("abcdef");

y.delete(1,100);//a java will delete even if the 100 index is not there java will assume that you made a mistake will compile

StringBuilder builder= new StringBuilder("pigeon dirty");

builder.replace(3,6,"sty");//pigsty dirty

StringBuilder builder2= new StringBuilder("pigeon dirty");

builder2.repalce(3,100,"");//pig this like delete

StringBuilder sb4= new StringBuilder("ABC");

sb4.reverse();

REMEMBER

String overrides equals(). It compares objects

StringBuilder don't overrides() . it compares references

StringBuilder compares the refrences but String compares values

StringBuilder one= new StringBuilder();

StringBuilder two= new StringBuilder();

StringBuilder three= one.append("a");

one==two;//false

one==three;//true coz the stringBuilder compares refrences

String x= "Hello world";

String z = " Hello world".trim();

sout(z);//Hello world trim() will remeove space at the begining and end

x.equals(z);//true this compares the values

Tiger t1= new Tiger();

Tiger t2= new Tiger();

Tiger t3=t1;

sout(t1==t3);//true coz objects are refrenece the same objects

sout(t1==t2);//false coz objects are referencing diferent objects in memory

sout(t1.equals(t2));

compare a String and StringBuilder

-you can not compare them coz they are of dirent types

Memory Allocation

1. Stack

- methods , locals variables and reference variables

- the first method loaded in is the main();

-String a =new string("kuu");

//this will create asn object into the heap then if String kuu is there in the string it will be refrenec by the object

-

2. Heap - objects/ string pool ( may also contains reference variables)

3. Static - Stataic data/methods

4. Code - bytecode

String pool - intern pool

- a location in JVM contains reusable literals values and constants

String name="Kudzie";

String name1= "Kudzie";

String name2= new String("Kudzie");

sout(name==name1);//true

sout(name==name2);//false

String name3="Kudz"+"ie";

String name4="Kudz";

name4+="ie";//created an new object by .concat()

sout(name==name3);//true

sout(name==name4);//false

String x="Hello world";

String y=" Hello world".trim();

sout(y);//this returns new string

sout(x==y);//false this referencing from difernt objects

String allKids="kudzai Kundai";

String firstKid="Kudzai";

firstKid += "Kundai";

sout(allKids==firstKid);//false coz += is used to firstKid=firstKid.concat("firstKid"); //calling a function will recreate an object again

string.intern(); //this checks in the String pool where the string is there if it is ther it returns the string if not there it create the string into string pool

String

- immutable, meaning its value cannot be changed once created.

-Every modification leads to a new object

- ensures thread safety but can be inefficient for frequent changes

-  immutable, it is inherently thread-safe

- Slower when it comes to frequent modifications due to immutability

- Stored in the String Pool, helping in-memory optimization.

-Concatenation creates new objects, leading to more garbage collection.

-Not suitable for extensive concatenation operations.

- Use when the text won't change, and thread safety is required.

**StringBuilder**

are mutable

**-** Not synchronized, and therefore not thread-safe. Faster for this reason

**-** Faster for repeated changes since no new objects are created.

**-** append method enables efficient concatenation without creating new objects.

**-** Use in a single-threaded environment for frequent changes.

**-**-introduced in version 1.0

**StringBuffer**

are mutable

- Synchronized, making it thread-safe.

-adds overhead and makes it slower than StringBuilder.

- Slightly slower than StringBuilder due to synchronization.

- append method enables efficient concatenation without creating new objects.

- Use in multi-threaded scenarios where text changes often.

-introduced in version 1.0

**Method Channing**

- common syntax for invoking multiple method calls in object-oriented programming languages

- practice of calling different [methods](https://www.geeksforgeeks.org/methods-in-java/) in a single line instead of calling other methods with the same object reference separately

class A {  
 private int a;  
 private float b;  
 A() { System.*out*.println("Calling The Constructor"); }  
 public A setint(int a) {this.a = a;return this;}  
 public A setfloat(float b) {this.b = b;return this;}  
 void display() {System.*out*.println("Display=" + a + " " + b);}  
}  
  
// Driver code  
public class Example {  
 public static void main(String[] args) {// This is the "method chaining".  
 new A().setint(10).setfloat(20).display();  
 }  
}

**LOCAL VARIABLE TYPE INTERENCE - var**

**Inferring the Type with var**

**-** keyword **var** can be when declaring local variables instead of the type i.e int,double,Long under certain conditions

- use this feature, you just type var instead of the primitive or reference type

-**var** is used to declare only local variable

public class VarKeyword {  
 var tricky = "Hello"; // DOES NOT COMPILE  
}

-variable defined within constructor, method or initializer block

public class Zoo {  
 public void whatTypeAmI() {  
 var name = "Hello";  
 var size = 7;  
 }  
}

**Type inference**

-specific type defined at the compile time

-**var** was introduced in java 10

-used to detects the data type of a variable based on the surrounding context

**Benefits**

1. A var is used as a local variable with initial value in

- constructor

String name="kudzie";  
var name2="kudzie";

-method

initializer block

{  
 double num=1.0;  
 var num=1.0;  
}

-loops

var size=7;  
 for(var i=0;i<size;i++){  
}  
for(var arg :args){  
  
}

-non denotable type

2. The value of a var can change but the type cannot change

Cannot do

1. A var can't declare without an initial value. It is always initialized on the same line ( or statement) where it is declared.

var question;  
question=1;//this will show compile error  
var answer;//cannot compile

you should declare and initialize in one line

var silly  
 =1;//this will compile java assumes it as one coz the statement is terminated with a semi colon

2. A var cannot be initialized with a null value without a type

var n=null;//cannot compile  
var o= (String) null;//compiles  
System.*out*.println(o);//null  
var s="java";  
s=null;//compiles  
var i=5;  
i=null;//cannot compile coz you cannot change the datatype of var

3. A var is not permitted in a multiple-variable declaration

Compound declaration

var a=2,b=10;//can't compiles  
int a, var c=7;//will not compiles coz all the types declared on a single line must be the same type and share the same declaration

4. A var cannot be used to initialize an array

double [] prices={9.3,10.3};  
var[] prices={9.3,10.3};//this will not compile

5. A var cannot be used as parameters, return types and fields

public class VarKeyword{  
 var name="java";//not allowed no declare instance or class variables  
 static var *counter*=0;//cannot compile coz var can be local not instance  
 public VarKeyword(var name){//var variable cannot be a parameter on the constructor  
 }  
 public var getName(){//var cannot be a return type  
 return name;  
 }  
 public void setName(var name){  
 this.name=name;  
 }  
}

6. A var is a reserved type name but not a keyword i.e can be used as an identifier expect as class, interface or enum name

public class Var{  
 public void var(){  
 var var="var";  
 }  
 public void Var(){  
 Var var=new Var();  
 }  
}

what we cannot do

class var{}//compile error  
interface var{}//compile error  
enum var{}//compile error

Var program examples

public void doesThisCompile(boolean check) {  
 var question;//will not compile coz the var variable needs to be intializedon the same line of declaration  
 question = 1;  
 var answer;//will not compile coz the var variable needs to be intializedon the same line of declaration  
 if (check) {  
 answer = 2;  
 } else {  
 answer = 3;  
 }  
 System.*out*.println(answer);  
}

**REMEMBER**

1. **Choose a variable name that provides useful information**

List<String> result = List.*of*("Kudzai","kudzanai","kundai");  
var result1 = List.*of*("Kudzai","kudzanai","kundai");//doesn’t give more information  
var stringList = List.*of*("Kudzai","kudzanai","kundai");

2. **Minimize the scope of local variable**

-use less lines of code so that you can easily follow you **var**

3. **Consider var when the initializer provides sufficient information**

var outputStram = new ByteArrayOutputStream();  
var reader = Files.*newBufferedReader*(Path.*of*(""));  
var list= List.*of*("a","b","c");

4**. Consider var to split chained or nested expression**

List<String> strings = List.*of*("Kudzai","kudzanai","kundai",""Hello","how","are");  
public Optional<String> getMaxEntry(){  
 return strings.stream()  
 .collect(*groupingBy*(s -> s, *counting*()))  
 .entrySet()  
 .stream()  
 .max(Map.Entry.*comparingByValue*())  
 .map(Map.Entry::getKey);  
}

for more readability change variable into **var**

public Optional<String> getMaxEntry(){  
Map<String,Long> collect= strings.stream()  
 .collect(*groupingBy*(s -> s, *counting*()));  
 var colect2= strings.stream()  
 .collect(*groupingBy*(s -> s, *counting*()));  
 Optional<Map.Entry<String, Long>> max=collect.entrySet()  
 .stream()  
 .max(Map.Entry.*comparingByValue*());  
 var max2=collect.entrySet()  
 .stream()  
 .max(Map.Entry.*comparingByValue*());  
 return max2.map(Map.Entry::getKey);  
 }

5. **Don't worry to much "programming to the interface" with local variables**

List<String> list=new ArrayList<>();  
list=new CopyOnWriterArrayList<>();  
list=new LinkedList<>();  
var list10=new ArrayList<>();  
list1=new CopyOnWriterArrayList<>();//doesn’t compile coz you can not access the methods coz you decare the variable as var  
list1=new LinkedList<>();

**6. Take care using var with diamond or generic methods**

List<String> list=new ArrayList<>();  
List<String> list2=new ArrayList<String>();//the left and right of |String type  
var list3=new ArrayList<>();  
var list4=new ArrayList<Object>();  
var list5=new ArrayList<String>();//compiles  
var list6=new ArrayList<List>();

**7. Take care var with literials *(Any constant value which can be assigned to the variable)***

list1=new CopyOnWriterArrayList<>();//doesn’t compile coz you can not access the methods coz you decare the variable as var  
list5=new LinkedList<>();  
var list3=new ArrayList<>();  
var list4=new ArrayList<Object>();  
var list5=new ArrayList<String>();//compiles  
var list6=new ArrayList<List>();

- No issue with Boolean, Character, long, String and float

boolean ready=true;  
char ch='\ufffd';  
long sun=0L;  
String label="kudzie";  
float f1=1.0f;  
double d1=3.5;  
  
char ch2='\ufffd'; //will be converted to boolean  
var sum=0L; //will be converted to long  
var label2="kudzie"; //will be converted to String  
var f12=1.0f; //will be converted to float  
var d12=3.5; //will be converted to double

whole numbers inferred as Integer

byte flags=0;  
short mask=0X7fff;  
long base 17;  
  
var flags2=0; //will be converted to int  
var mask2=0X7fff; //will be converted to int  
var base3=17; //will be converted to int

**SOLUTION**

- you have to cast the right-hand side

var flags2=0; //will be converted to int  
var mask2=0X7fff; //will be converted to int  
var base3=17; //will be converted to int  
  
var flags3=(byte) 0;  
var mask3=(short) 0X7fff;  
var base4=(long) 1;//OR  
var long3=1L;

**Managing Variable Scope**

-local variables are declared within a code block.

How many variables do you see that are scoped to this method?

public void eat(int piecesOfCheese) {// bitesOfCheese variable is declared insidethe method.   
 int bitesOfCheese = 1;//The piecesOfCheese variable is a method parameter.  
}  
}

There are two variables with local scope

-Neither variable can be used outside of where it is defined.

**Limiting Scope**

-Local variables can never have a scope larger than the method they are defined in

public void eatIfHungry(boolean hungry) {  
 if (hungry) {  
 int bitesOfCheese = 1;  
 } // bitesOfCheese goes out of scope here  
 System.*out*.println(bitesOfCheese); // DOES NOT COMPILE coz the variable is out of scope  
}

-The variable hungry has a scope of the entire method, while the variable bitesOfCheese has a smaller scope

-When you see a set of braces ({}) in the code, it means you have entered a new block of code

-Each block of code has its own scope

-t blocks can contain other block.

-smaller contained blocks can reference variables defined in the larger scoped blocks, but not vice versa

public void eatIfHungry(boolean hungry) {  
 if (hungry) {  
 int bitesOfCheese = 1;  
 {  
 var teenyBit = true;  
 System.*out*.println(bitesOfCheese);  
 }  
 System.*out*.println(teenyBit); // DOES NOT COMPILE teenyBit has a limited scope  
 }  
 System.*out*.println(teenyBit); // DOES NOT COMPILE teenyBit has a limited scope  
 }

Tracing Scope

public void eatMore(boolean hungry, int amountOfFood) {  
 int roomInBelly = 5;  
 if (hungry) {  
 var timeToEat = true;  
 while (amountOfFood > 0) {  
 int amountEaten = 2;  
 roomInBelly = roomInBelly - amountEaten;  
 amountOfFood = amountOfFood - amountEaten;  
 }  
 }  
 System.*out*.println(amountOfFood);  
 }

-the above program compiles

**figuring out the scope**

-is to identify the blocks of code i.e 3blocks

-look at the scope of each variable

**Applying Scope to Classes**

Instance variables

- states that they are available as soon as they are defined and last for the entire lifetime of the object itself.

-In scope from declaration until the object is eligible for garbage collection

Static variables

-they stay in scope for the entire life of the program

-In scope from declaration until the program ends

public class Mouse {  
final static int *MAX\_LENGTH* = 5;//static variables needs to be intialized at the same line of declaration or in the constructor coz of the final keyword  
int length;//instance variables with default value of 0  
public void grow(int inches) {  
 if (length < *MAX\_LENGTH*) {  
 int newSize = length + inches;//newSize and inches are local variable  
 length = newSize;  
 }  
 }  
}

**LAMBDA EXPRESSION**

- assigning a function into a variable

*Function<Integer,Integer> f=(var num) -> 1;*

*var f=(var num) -> 1;* *//does not compile coz the compiler does not know what is the data type of the rightside*

**SOLUTION**

*var f=(Function<Integer,Integer>) (var num) -> 1;*

**BENEFITS**

lambada expressions can also be implicitly typed with var syntax

*(var num) -> 1*

var can be used with annotations

*(@Notnull var a, var b) -> "Hello"*

*BiFunction<Integer,Integer, String> f2=(@Notnull var a, var b) -> "Hello";*

var can be used for **non -denotable** (anonymous /nameless/unknown) class or intersection

**Non Denotable** means you can use the variables of Anonymous class outside that class becoz of var

//Anonymous class

Object productInfo = new Object(){

String name= "Bannana";

int total=0;

};

//Outside the Anonymous class

sout("name"+productInfo.name+"+productInfo.total);// will not compile coz the variables name are declared and initialized inside the Anonymous class

**SOUTION** i.e Getting data from outside of an anonymous block  
*var productInfo = new Object(){*

*String name= "Bannana";*

*int total=0;*

*};*

*System.out.println("name "+productInfo.name+", total "+productInfo.total);*

Output:  
name Bannana, total 0  
  
**INTERSECTION TYPE**(Function<Integer,Integer> & Serializable) action = (Function<Integer,Integer> & Serializable) i -> i+1;// does not compile  
  
**SOLUTION**   
-developers they split

*Function<Integer,Integer> f = Function<Integer,Integer> i -> i+1;*

Serializable s = Serializable i -> i+1;

or with the use of var

var action = (Function<Integer,Integer> & Serializable) i -> i+1;

**CANNOT DO**

-you can't mix var and non -var parameters

*(var a, Integer b) -> true; // cannot compiles*

**SOLUTION**

*boolean y=(var a, var b) -> true; //compiles when one is var all should var*

*System.out.println("x : "+ x);*

*Boolean b=(String x, var y, Integer z) -> true;*

*System.out.println("b : "+ b); //will not compile coz you can’t mix var and primitive or with Wrapper classes* **SOLUTION**

*boolean y=* (var x, var y, var z) -> true;

*System.out.println("y : "+ y); //true*

-you cannot mix var and explicit type parameters

double d=(var b, var k, var m) -> 3.14159; // compiles

(var x,y)-> "goodbye //doesn’t compile

**SOLUTION**

*(var x,var y)-> "goodbye*

-you can't ommit parenthesis for single var parameter

*var w -> 99*

**SOLUTION**

*(var w) -> 99*

**JAVA OBJECT ORIENTED APPROACH**

-declare objects

-instantiate objects including nested class objects

-explain objects lifecycle (including creation, dereferencing by reassignment and garbage collection)

**OBJECT**

-objects are runtime instances in the memory

-variables and fields names are properties/ state of the objects

-stored in the heap memory

**DECLARATION**

Random r

-this the object reference that points an object in a memory

**INSTANTIATION**

=new

-new keyword used to create the object into a memory

**INTIALIZATION**

*Random();*

*Random r;*

*r=new Random();*

*sout(r.nextInt(10))*

**TOP LEVEL TYPE / OUTER LEVEL (CLASS, INTERFACE, ENUM)**

-note defined within another type declaration

-can only set with *public* or *package-private access (default)*

-java file may have at most one public top-level type and must match the name of the file

**INNER LEVEL TYPE**

**NESTED CLASS**

-class that is defined within another type

static nested class

-static type defined at the memebver level of a classs

-nesting creates a namespae becoz the enclosing class name must be used to refer to it

-can be private or use one of the other access modifiers to encapsulate it

-enclosing class can refer to the fields and methods of static nested class

>Enclosing.java

public class Enclosing{

/\*public , private, pubublc \*/ static class Nested{

private int price=6;

private static double=8.0;

}

psvm(){

//Accessing variable from inner nested class

sout(Enclosing.Nested.price);

sout(Nested.price);

Nested nested = new Nested();

nested.rate);

}

Inner nested class

-non static type defined at the memeber level of a class

public class Outer{

protected class Inner{

public void go(){}

}

public void callInner(){

new Inner.callInner();}

psvm(){

Outer outer = new Outer();

outer.callInner();

or

Inner inner = outer.new Inner();

inner.go();

}

}

-can be declared public , protected, package-private

-can extend any class and implements interfaces

protected class Inner Implements Main(){}

-can be marked abstract or final

protected abstract class Inner{}

protected final class Inner{}

-can not static declare fields , except for static final fields

- can access members of the outer class including private methods

class A{

private int x=10;

class B{

private int x=20;

class c{

private int x=30;

public void allTheX(){

sout(x);//30

sout(this.x);//30

sout(B.this.x);//20

sout(A.this.x);//10

}

}

}

psvm(){

A a = new A();

B b= a.new B(); or A.B b= a.new B();

B.C c b.new C(); or A.B.C c b.new C();

c.allTheX();

}

}

Local nested class

-class defined within a methody body

-donot have an access modifier

-cannot be declared static and cannot declare static fields or methods expect for static final

-have acess to all fields and methods on the enclosing class ( when defined in an instance method)

-can access local variables if the variables are final or effectively final

>PrintNumbers.java

public class PrintNumbers{

private int length=5;

public void calculate(){

final int width=20;

int height=6;//this variable is effectively final othe calculate() and as soon as you resign the varible you get compile error

class MyLocalClass{ //can't declared as public , protected, private and static

public void multiply(){

sout(length\*width));

}

MyLocalClass local = new MyLocalClass();

local.multiply();

}

psvm(){

PrintNumbers pn = new PrintNumbers();

pn.calaculate();

}

}

Anonymous nested class

-special case of the local class that does not have a name

- ca't be declared static , final and abstract

-can't extends other than object class or existing interface

-can't have a constructor

-can be defined where they are needed

-they have access to all fields and methods of the enclosing class

-they can access local variables if the variable are final or effectively final

>ZooGiftShop1.java

public class ZooGiftShop1{

abstract class SaleTodayOnly{

abstract int dollarsOff();

}

public int admission(int basePrice){

new SaleTodayOnly(){//Anoynmous class- this is an extension of the abstract class SaleTodayOnly and also the Anonmous function is the child of abstract class

@Override

int dollarsOff(){

return 3;

}

};

or we can assign Anonmous function into reference variable

SaleTodayOnly sale=new SaleTodayOnly(){

@Override

int dollarsOff(){

return 3;

}

};

return basePrice - sale.dollarsOff();

}

}

with inteface

>ZooGiftShop2.java

public class ZooGiftShop2{

interface SaleTodayOnly{

int dollarsOff();//by default the method is public abstract

or public abstract int dollarsOff();

}

public int admission(int basePrice){

SaleTodayOnly sale=new SaleTodayOnly(){

//SaleTodayOnly();//doesnot need contructor will not compile

@Override

int dollarsOff(){

return 3;

}

};

return basePrice - sale.dollarsOff();

}

}

Defining an anonymous function as a method parameterwhere we need it

public int pay(){

int local=10;//this varible is effectively final i.e final local if you change the varible literal results in compile error

return admission(5, new SaleTodayOnly(){

@Override

int dollarsOff(){

return 3;

}

});

}

Modifiers to nested classes

Permited modifiers static nested class inner class local class Anonymous class

Access modiffers all all none none

abstract yes yes yes no

final yes yes yes no

Memebers in nested classes

Permited Memebers static nested class inner class Local class Anonymous class

Instance methods yes yes yes yes

instance variables yes yes yes yes

static methods yes yes yes yes

static variables yes yes yes yes

(if final)(if final) (if final)

Nested class access rules

Can extends any class or implement any number of interface yes yes yes no - must have axactly one superclass or one interface

can access instance members of enclosing class without a reference ? no yes yes(if declared in an instnace method) yes(if declared in an instance method)

can access local variables of enclosing method ? N/A N/A yes(if final or effective final) yes(if final or effectively final)

Nested class

>static non static nested

inner local Anoynmous

Memory allocation

1. Stack

-methods, local variables, and refrenece variables.

-reference may or may not be created on the heap

-all reference are same size no matter the data type and accessed by thir variable names

2. Heap

-objects ( may contain reference variables)

- is the dynamic memory that can change size

-also called free memory

-can controls deallocation

-stores string pool and static area

-have no name in the heap can only be accessed by reference

-objects may vary in size depending to the class defination

3. Static or global

-static data/methods

-variables that are accessable allover the code

4. Code

-byteode

Difference btwn Heap and stack

Stack Heap

Storage methods local variables Objects

,reference to objects instance variables

Order LIFO Memory mngment techniques

life Current method application

size smaller No size limit

efficiency faster slower

Allocation/ Allocated-method call Alocated-new

Deallocation deallocated-method return Deallocated-Garbage collector

memory is full java.lang.StackOverflowError java.lang.OutOfMemoryError

Garbage Collection

-automatically freeing memory on the heap by deleting objects those are no longer reachable/ or needed by the program

-JVM is responsibility to do garbage collection automatically.

-Java provides a garbage collector to automatically look for objects that aren’t needed anymore.

-All Java objects are stored in your program memory’s heap

-As a developer, the most interesting part of garbage collection is determining when the memory belonging to an object can be reclaimed

-garbage-collection eligibility like shipping a package which can fail to deliver or collect the postal for delivery coz of holiday or bad weather

-Java includes a built-in method to help support garbage collection where you can suggest that garbage collection run.

System.gc();

Garbage collector

-automatically look for objects that are not need anymore

Tracing eligibility for garbage collection

-object state of no longer reachable when

- the object no longer has any reference pointing to it

-all reference to the object has gone out of scope

System.gc();

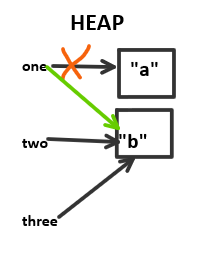
-suggest JVM to kill off garbage collector

-is not guaranteed to run or do anything

|  |  |
| --- | --- |
| Objects | References |
| -object sits on the heap and does not have a name. | - a variable that has a name |
| -no way to access an object except through a reference | - can be used to access the contents of an object |
| -An object cannot be assigned to another object, and an object cannot be passed to a method or returned from a method | -A reference can be assigned to another reference, passed to a method, or returned from a method |
| -. Objects come in all different shapes and sizes and consume varying amounts of memory | -. All references are the same size, no matter what their type is. |
| object that gets garbage collected | References are garbage collected |

When each object is going to be garbage collected ?

public class Scope {  
public static void main(String[] args) {  
 String one, two;  
 one = new String("a");  
 two = new String("b");  
 one = two; //one object that content "a" is eligible for garbage collection  
 String three = one;  
 one = null;//two object that content "b" is eligible for garbage collection  
}//two & three will eligible for garbage collection when they’re out of scope that the outside of the main() method block  
}



Object

finalize()

-can run zero or one times. it can not run twice

@Override

protected void finalize() throws Trowable{

super.finalize();

}

Interface

-Create and use interfaces,

-identify functional interfaces,

-utilize private , static and default methods

Since java version Membership type equired modifiers Implicity modifiers Has a value or body

constant 1.0 class - public,static,final yes

variable

Abstract 1.0 instance - public,abstract no

method

Default 8 instance default public yes

method

Static 8 class static public yes

method

Private 9 instance private - yes

method

Privte 9 class private,static - yes

static

method

> InterfaceSuamamary.java

public interface InterfaceSummary{

/\*public static final \*/ int CONSTANT =1;

/\*public abstract\*/ int abstractMethod();//public abstarct - implicity declared by the compiler if it is not there

/\*public\*/ default int defaultMethod(){//this the concrete method

abstractMethod();//can call abstract method

privateMethod();

staticMethod();

return CONSTANT;

}

private int privateMethod(){

abstractMethod();

defaultMethod();

staticMethod();

return CONSTANT;

}

/\*public\*/ static int staticMethod(){

privateStaticMethod();//can only call static method directly

return CONSTANT;

}

private static int privateStaticMethod(){

return CONSTANT;

}

}

Instance Method in Interface

-can access static method

abstarct

default

Non static private

Class/Static method in interface

-can't access instance method

static

static private

colum 1

Acessible from default and private methods within interface defination

column 2

Acessible from static methods within the interface defination

column3

Acessible from instance methods implementing or extending the interface

column4

Accessible outside the interface without an instance of interface

constant yes yes yes yes

varible

abstract yes no yes no

method

default yes no yes no

method

private yes no no no

method

static yes yes yes yes

method

private yes yes no no

static

method

Example

WithInterface.java

public class WithInterface implements InterfaceSummary{

@Override

public int abstarctMethod(){

defaultMethod();

//abstarctMethod(); // can be also accessed

interfaceSuammary.staticMethod();

return CONSTANT;

//you can not access private method and variables

}

}

WithoutInterface.java

public class WithoutInterface{

public int instantMethod(){

//can only access static methods & variables

InterfaceSummary.staticMethod();

return InterfaceSummary.CONSTANT;

}

}

Let says you have multiple interface with same methods

Bob.java

instrface Walk{

public default int getSpeed(){

return 10;

}

}

instrface Speed{

public default int getSpeed(){

return 2;

}

}

public class Bob implements Walk, Read{

//class Bob is now confused which method (getSpeed() ) to implement since there are all same

@Override

public int getSpeed(){

return 0;

}

//Solution - to avoid confusing the compiler

public int getWalkSpeed(){

return Walk.super.getSpeed();

}

psvm(){

System.out.printIn(new Bob().getSpeed());//0 from the class

System.out.printIn(new Bob().getWalkSpeed());//10 from the interface

}

}

Reason for the private methods

-avoid the code duplication and encapsulation

Function interface

-contains a single abstract methods (SAM)

-excluding any public method already defined in the java.lang.Object class

-cannot decalre an interface method that is incompatible with object.

-for example int toString()

Lambda expression

-a block of code that gets passed around, sort of like an nonymous class that defines one method

-any function interface can be implemented as a lambda expression

Defining functional interface

Lion.java

@FunctionalInterface//this will gives you a compile if the interface is not function but its optional

interface Eat{

void eat(int amountOfFood);

//methods that are counted as non function interface method even if you define the interface is still valid coz this methods came from java.lang.Object or specificaly from String class

-or the following methods don't count in SAM

String toString();

boolean equals(Object obj);

int hashCode();

}

public class Lion implements Eat{

@Override

public void eat(int amountOfFood){

sout(amountofFood);

}

}

More example of functional interface for clarity

Sprint.java

@FunctionalInterface //it only have one method

public interface Sprint{

void sprint(int speed);

}

@FunctionalInterface

interface Dash extends Sprint{}

//his is not a function interface coz its inheriting another interface so it has two method

interface Skip extends Sprint{

void skip();

}

//is not functional interface coz it doesn't have an abstract method

interface sleep{

private void snore(){}

public default int getZzz(){ return 1;}

}

@FunctionalInterface //coz it only have one abstract method

interface climb{

void reach();

default void fall(){}

static int getBackup(){return 100;}

private static boolean checkHeight(){

return true;

}

}

Create and use enumerations

Enum

-enumerated types

-fixed set of constants

-is the type of class that mainly contains CONSTANTS (List of values) and helper methods

-it implicitly extends from abstarct class java.lang.Enum

-each enum class will entends the following methods

protected Object clone()//object

int compareTo(E o) //compares this enum with specific object for order

Optional<Enum, EnumDes<E>> describeConstable()

boolean equals(Object other) //returns true if the specified object is equal to this enum constant

protected void finalize() //enum class cannot have finalize methods

Class<E> getDeclaringClass() //returns the class object corresponding to this enum constant's enum type

int hashCode() //returns hasCode for the enum constant

String name() //returns the name of this enum constant, exactly as declared in its enum declaration

int ordinal() // returns the ordinal of this eni=umeration constant(its position in its enum declaration, ehere the intial constant is assigned as ordinal zero)

String toString() //returns the name of this enum constant as contained in the declaration

static <T extends Enum<T>> valueOf(Class<T> enumType, String name) //returns the enum constant of the specified enum type with the specified name

-it implicity final i.e it can not extends but implements interfaces

-it provides type safe checking . Invalid enum value introduce compile error

Day.java

interface DayIntrface{}

public enum Day{// implements DayInterface//the enum is implicity final

int i=0;//show compile error

SUNDAY("High"){//method overriding

public void printTemp(){

sout("Temp is 48deg");

}

},

MONDAY("High"),

TUESDAY("High"),

WEDNESDAY("High"),

THURSDAY("High"),

FRIDAY("Low"),

SATURDAY("High");//put a semi colon if you want to declare methods and variables

private String temp;

/\*private\*/ Day(String temp){//private is the default access modifier its optional to ddefine it coz its implicitly define

-the constractor is creted once just like class constractor

this.temp=temp;

}

public void printTemp(){

sout("Temp is 25deg");

}

public abstract void recordTemp();//each enum value is required to override an abstarct method

}

public ExtendedEnum extends Day{//cannot extends a Enum this can not compile

}

calling enum

Day d= Day.TUESDAY;//compile attach two methods valueOf

Day d1 = Day.valueOf("TUESDAY");

d1 = Day.valueOf("tuesaday");//this will result in the runtime error i.e No enum constant

we can use == or equals to compare two enums

sout(d==d1);//true

sout(d.equals(d1));//true

sout(Day.SUNDAY.toString());//Day.SUNDAY is an object

for Day day: Day.values()){//values(); returns all the enum as an array

sout(day.name+"---"+day.ordinal());

name() vs .toString()

-final -is not final

can not override -can be override

output is same -same output

if(Day.TUESDAY==4){//if you use invalid enum it will show compile error to show that it is type safety

}

Calling enum methods

d.printTemp();//25deg

d.SUNDAY.printTemp();//48deg the override method of SUNDAY

}

creating a switch

switch(d){

case SUNDAY:

System.out.println("Go to beach");

break;

case MONDAY://Error Day.TUESDAY

System.out.println("Go back to work");

break;

default:

sout("Are you at work");

}

Remember

-Values:

-are always decalred first otherwise it will show compile error

-end with a semi-colon (;) if there is anything beside values

-a non final enum method can be overridden by any enum value

- required to implement abstract method

HELPER METHODS

-enum methods - compiler provided two public static methods

-valueOf(String name) - returns the enum value

-values() - get a array of all values

Overrided method

toString() or name() -it prints the name of the enum

equals() or == it compares the enums

ordinal() - corresponding index value

clone() - guarantees enums are never get cloned

Constructor

-implicitly private

-called once in the beginning to create enum values. After that Java just returns the already constructed enum values

Compiler Error

-Code will not compile if

- values- are not declared first

-constructor - contains public or protected modifier

-Switch block - contains case as Enum.TYPE

-extend an enum

-compare enum value with integer

**WORKING WITH ARRAYS AND COLLECTION  
GENERICS**

-type parameter for code

-provides compile type safety  
- means parameterized types  
-allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods

**GENERIC COLLECTION**

-contain specific type

-disable the type-casting and there is no use of type-casting when it is used in generics

- type-safe and checked at compile-time not at runtime

-allow the data-types to pass as parameters to classes

**NON GENERIC COLLECTION**

-contains any thing

**NB**

The classes, which accept one or more parameters, ​are known as parameterized classes or parameterized types.

>LegacyCode.java

static void printName(List list){

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

String name =(String) list.get(i);//Runtime Error the String Builder cannot be casted to String

sout(name);

}

}

psvm(){

List names= new ArrayList();

// List<String> names= new ArrayList<String>();

names.add(new StringBuilder("kudzie"));//will show compile time error coz you cannot add A StringBuilder to a String ArrayList

printNames(names);

}

**GENERIC CLASS**

ClassName.java

public class ClassName<T>{

}

class Type Parameter<T> : generic data type

-diamond operator <> : shorthand notation allows to omit the generic from the right side

Create.java

public class Create<T>{

private T contents;

//getter & setter method

}

>Travel.java

class Elephant{}

class Zebra{}

public class Travel{

psvm(){

Elephant elephant = new Elephant();

Create<Elephant> createForElephant = new Ceate<>();

createForElephant.setContents(elephants);

Elephant inNewHome = createForElephant.getCentents();

Zebra zebra = new Zebra();

Create<Zebra> createForZebra = new Ceate<>();

}

}

Declaring the generics

java 5

*List<Integer> list = new ArrayList<Integer>();*

*Map<String, Integer> map= new HashMap<String,Integer>();*

*Map<Long, List<Integer>> listMap= new HashMap<Long, List<Integer>>();*

-List is the interface and ArrayList is a concrete class

java 7

- we can ommit the right side just use empty diamond operator

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

Map<String, Integer> map= new HashMap<>();

Map<Long, List<Integer>> listMap= new HashMap<>();

**NAMING CONVENTION**

E- element

K - map key

V - map value

N - number

T - generic data type

S, U, V - multiple generic types

>SizeLimitedCreate.java

*public class SizeLimitedCreate<T,U>{*

*private T contents;*

*private U sizeLimit;*

*}*

>Travel.java

*psvm(){*

*Integer numPonds= 15\_000;*

*SizeLimitedCreate<Elephant, Integer> slc = new SizeLimitedCreate<>();*

*}*

**TYPE ERASURE**

-the process of removing generic syntax for code

-can occur at class (or variable) and method levels.

-ensures type safety of our code and prevents runtime errors.

-process of enforcing type constraints only at compile time and discarding the element type information at runtime.

-it provides backward compatibility

-Replace all type parameters in generic types with their bounds or Object if the type parameters are unbounded.

-the compiler discards the type parameters on the class and replaces them with its first bound, or Object if the type parameter is unbound.

*public class Stack<E> {*

*private E [] stackContent;*

*public Stack(int capacity) {*

*this.stackContent = (E[]) new Object[capacity];*

*}*

*public void push(E data) {*

*// ..*

*}*

*public E pop() {*

*// ..*

*}*

Upon compilation, the compiler replaces the unbound type parameter E with Object:

*public class Stack {*

*private Object[] stackContent;*

*public Stack(int capacity) {*

*this.stackContent = (Object[]) new Object[capacity];*

*}*

*public void push(Object data) {*

*// ..*

*}*

*public Object pop() {*

*// ..*

*}*

In a case where the type parameter E is bound:

*public class BoundStack<E extends Comparable<E>> {*

*private E[] stackContent;*

*public BoundStack(int capacity) {*

*this.stackContent = (E[]) new Object[capacity];*

*}*

*public void push(E data) {*

*// ..*

*}*

*public E pop() {*

*// ..*

*}*

*}*

compiler will replace the bound type parameter E with the first bound class, Comparable in this case:

*public class BoundStack {*

*private Comparable [] stackContent;*

*public BoundStack(int capacity) {*

*this.stackContent = (Comparable[]) new Object[capacity];*

*}*

*public void push(Comparable data) {*

*// ..*

*}*

*public Comparable pop() {*

*// ..*

*}*

*}*

**METHOD TYPE ERASURE**the method's type parameter is not stored but rather converted to its parent type Object if it's   
  
unbound or it's first bound class when it's bound.

-it replace the type parameter with parent type

*public static <E> void printArray(E[] array) {*

*for (E element : array) {*

*System.out.printf("%s ", element);*

*}*

*}*

Upon compilation, the compiler replaces the type parameter E with Object:

*public static void printArray(Object[] array) {*

*for (Object element : array) {*

*System.out.printf("%s ", element);*

*}*

*}*

For a bound method type parameter:

*public static <E extends Comparable<E>> void printArray(E[] array) {*

*for (E element : array) {*

*System.out.printf("%s ", element);*

*}*

*}*

We'll have the type parameter E erased and replaced with Comparable:

*public static void printArray(Comparable[] array) {*

*for (Comparable element : array) {*

*System.out.printf("%s ", element);*

*}*

*}*

what will happen behind the sence

-the compiler will convert T to Object

**GENERIC INTERFACE**

1. Specific generic type in the class

2. Create a generic class

3. Raw type - Not use generic at all

>ShippableRobotCreate.java

*//Option 1*

*class Robot{}*

*public interface Shippable<T> {*

*void ship(T t);*

*}*

*public class ShippableRobotCreate implements Shippable<Robot>{*

*@Override*

*public void ship(Robot robot){}*

*}*

*}*

//option 2

*class Shippable AbstractCreate<U> implements Shippable<U>{*

*@Override*

*public void ship(U u){*

*}*

*}*

//option 3

*public class ShippableCreate implements Shippable{*

*@Override*

*public void ship(Object o){}*

*}*

*}*

**Can’t**

-call a constructor

*private T contents;*

*public ClassName(){//constructor*

*contents= new T();//compile error*

*array = new T[4];//can't compile error*

*}*

Create an array of that generic type

*private T[] array;*

*public ClassName(){ //constructor*

*array = new T[4]; //can't compile error*

*}*

-call instanceof

*if(T instanceof U){*

*sout("Not allowed");*

*}*

-use a primitive type as a generic type parameter

*List<int> list =new Arraylist<>();*

-create a static variable as a generic type parameter

*private static T constant;*

**GENERIC METHOD**

*public <T> List<T> getList(T t){*

*return new ArrayList<T>();*

*}*

-method type parameter<T>

-it specified before return type

- type parameter<T> on the method it always specified in static method since they aren't part of instance methods that can declare the type

*public static<T> void prepare(T t){}*

-it is optional in instance methods if they can obtain from the class/ interface

*// public T prepare(T t){return T;}*

-it is independent of class/ interface type parameter

-return type List<T>

-parameter T t

>Handler.java

*public class Handler{*

*//static methods*

*public static <T> void prepare(T t){*

*sout("preparing ..."+t);*

*}*

//public static <T>return type<T>

*public static <T>Create<T> ship(T t){*

*sout("Shipping ..."+t);*

*return new Create<T>();*

*}*

*psvm(String [] args){*

*ship("package");*

*ship(args);*

//or

*Hander.<String>ship("box");*

*Handler.<string[]>ship(args);*

*}*

*}*

- the class should provide a T for instant methods

*class More<T>{*

//static method - should provides its own T

*public static T staticMethod(T t){//compile error you have to provide T b4 return type*

*return t;*

*}*

public static <T> T staticMethod(T t){//compile error

return t;

}

//instance methods

public T instanceMethod(T t){return t;}

}

Bounded Parameter Type

-generic type that specifies a bound for a generic

Wildcard(?)

- (?) represent the unknow generic type

1. Unbounded<?>

- any type

-Immutable

- assign any type on he right side

2. Upper bounded<? extends TypeName>

- any type that is of typeName or extends it

-immutable

- assign OwnType, Object, parent or child classes on he right side

-assign owntype and its subtypes types

3. Lower bounded<T super TypeName>

-any type that is of TypeName or a subclass

- assign OwnType,Object, parent or child classes on he right side

-assign owntype and its supper types

>Main.java

class GrandParent{}

class Parent extends GrandParent{}

class Child extends Parent{}

public class Main{

//Generic - Exact math right or left side

List<GrandParent> obj =new ArrayList<GrandParent>();

//Grandparent to parent

List<GrandParent> obj = new ArrayList<Parent>();//Compile error GrandParent doent Macth with Parent

Solution

//unbonded

List<?> obj1 = new ArrayList<Parent>();

obj1= new ArrayList<Object>();

obj1=new ArrayList<GrandParent>();

obj1=new ArrayList<Child>();

cannot add data to a list i.e immutable

obj1.add(new Object());

obj1.add(new Parent());

obj1.add(new Child());

UpperBounded

List<? extends Grandparent> obj3 = new ArrayList<Parent>();

obj3= new ArrayList<Child>();

cannot add data to a list i.e immutable

obj2.add(new Parent());

obj2.add(new GrandParent());

obj2.add(new Child());

Lower bounded

List<? super Parent> obj4 = new ArrayList<GrandParent>();

obj4=new ArrayList<Parent>();

obj4 =ArrayList<Child>();

obj2.add(new Parent());//vpmpilre

obj2.add(new Child());

Normally the exam asks about methods in generic

MyClass.java

interface SuperInterface{}

interface Interface extends SuperInterface{}

public class MyClass implements Interface{

psvm(String [] args){

List<Interface> test1= new ArryList<Interface>();

//Upper bounded

List<? extends Interface> test2=new ArryList<Interface>();

test2 = new ArryList<MyClass>();

test2 = new ArryList<SuperInterface>();//will not compile coz SuperInterface is the super interface

//Lower bounded

List<? super MyClass> test3= new ArryList<MyClass>();

test3 = new ArryList<Interface>();

test3= new ArryList<SuperInterface>();

}

}

List<?> x1= new ArrayList<>();//list of objects

var x1= new ArrayList<>();//list of objects

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

numbers.add(Interger.valueOf(42));

List<Object> objects =numbers;//compile error

//objects.add(" forty two");will not compile

sout(numbers.get(0));

Upper bounded

List<? extends Number> list1 = new ArrayList<Integer>(Array.asList(1,2,3,4,5));

//type erasure

// List list1 = new ArrayList<Integer>(Array.asList(1,2,3,4,5));

sout(total(list1));

private static long total(List<? extends Number> list){

//private static long total(List list){

long count = 0;

for(Object object : list){

count+= number.longValue();

}

for(Number number : list){

Number number = (Number) object;

count+= number.longValue();

}

return count;

}

Lowerbound

LowerBounded.java

public class LowerBounded{

psvm(String [] args){

List<String> strings = new ArrayList<>();

strings.add("kudzabnai");

List<Object> objects = new ArrayList<>(strings);

addSound(strings);

addSound(objects);//this will results in the compile error coz object is not a string

}

private static void addSound(List<String> list){

list.add("kundai");

}

Solution

private static void addSound(List<? extends String> list){//allows String and Object as super class

list.add("kundai");

}

Flying.java

interface Flyer{

void fly();

}

class HangGlider implements Flyer{

@Override

public void fly(){

}

}

class Goose implements Flyer{

@Override

public void fly(){

}

}

public class Flying{

private void anyFlyer(List<Flyer> flyer){}

private void groupOfFlyer(List<? extends Flyer){//OwnType and its subclasses}

psvm(String [] arags){

Flying userFlyer = new Flying();

List<Flyer> flyers = new ArrayList<>();

userFlyer.anyFlyer(flyers);

userFlyer.groupOfFlyer(flyers);

List<Goose> goose = new ArrayList<>();

userFlyer.anyFlyer(goose);//compile error Goose is the subclass of Flyer

userFlyer.groupOfFlyer(goose);

}

}

NB this Lower/Upper Bounded also applies to interfaces just like classes

UnBonded are immutable

private static void addSound(List<?> list){

list.add("kundai");//UnBonded are immutable i.e cannot be added

}

//Upper Bonded are immutable i.e cannot be added

private static void addSound(List<? extends Object> list){

list.add("kundai");

}

//

private static void addSound(List<Object> list){

list.add("kundai");//compile error string cannot be added to the Object

}

//Lower bounded

private static void addSound(List<? super String> list){

list.add("kundai");//compiles coz this can hold OwnType(String) and SuperType(Object)

}

Remember

Can

- <TypeName> hold exact match

- wildcard<?> hold any match

<? extends TypeName> holds ownType and subTypes

<? super TypeName> holds own type and supertypes

- wildcards <?> only specify in variable declaration and method parameter

cannot

- add an item to the list with unbonded or upper bounded but you can add on the lower Bounded

class A{}

class B extends A{}

class C extends B{}

//UnBounded

List<?> list1= new ArrayList<>();

list1.add(new A());//compile error

//UpperBounded

List<? extends A> list1= new ArrayList<>();

list1.add(new A());//compile error

//Lower Bounded

List<? super B> list1= new ArrayList<>();

list1.add(new A());//this will compile

- use objects in the methods if type parameter and class in scope have same name

<B extends A>B fourth(list<B> list){

return new B();//will not compiles B is the parameter within the scope and the class is hidden withi the method

}

- specify bounds without wilcards(?)

void fifth(List<X extends B> list){//will not compile coz you cannot replace wildcard <?> by X

}

>Summary.java

class A{}

class B extends A{}

class C extends B{}

public class Summary{

psvm(String [] arags){

List<A> list = new ArrayList<A>();

list =new ArrayList<>();

List<?> list1= new ArrayList<>();

List<?> list1= new ArrayList<>();

List<? extends A> list2= new ArrayList<>();//can assign A,B,C and can compile

List<?> list1= new ArrayList<>();

List<? super A> list1= new ArrayList<>();// can assign only A and object

List<? super B> list1= new ArrayList<>();//A, Object

List<? super B> list1= new ArrayList<>();//B, A, Object

<T> T first(List<? extends T> list){//T, Object

return list.get(0);

}

void second(List<? super B> list){//T, Object

return list.get(0);

}

Will not compile

<T> <T> first(List<? extends T> list){//T, Object

return list.get(0);

}

}

}

Handle Exception

Exception

-java used exception to handle errors and other exceptional events

-it alter the flow of program

-java.lang.Throwable is theparent class of all exception

Types of Exception

1. Checked exception

-must be declared or handled by the application where it is thrown

e.g

IOException

FileNotFoundException

ObjectStreamException

NonSerialiableException

ParseException

SQLException

>will not compile

public class Main{

public static void main (String[] args){

new Main().scope();

}

public void scope(){

throw new FileNotFoundException("testing runtime exception");

//must be caught or declared to be thrown

}

}

2. UnChecked Exception

-doesn't need to be declared or handled by the application where it is thrown

>will compile

public class Main{

public static void main (String[] args){

new Main().scope();

}

public void scope(){

throw new RuntimeException("testing runtime exception");//will compile

}

}

Error

-Exception thrown by JVM

-they should not handled or declared

e.g

OutOfMemoryError

StackOverflowError

NoClassDefFoundError

Runtime Exception

-occurs while program is running

e.g

NullPointerException

ArrayIndexBoundException

ArithematicException

ClassCastException

IllegalArgumentException

NumberFormatException

TryArithematicException.java

public class TryArithematicException{

psvm(String... a){

int a=0;

int b=10;

int c = b/a;

sout(c);

}

}

TryArryIndexOutOfBoundException.java

public class TryArryIndexOutOfBoundException{

psvm(String... a){

int[] arrayElements= new int[10];

sout(arrayElements[-1]);

}

}

TryClassCastException.java

public class TryClassCastException{

psvm(String... a){

String type="kudzie";

//Integer number= (Integer) type;

Object obj=type;//coz everything comes from object

Integer number=(Integer) obj;

sout(number);

}

}

TryNullPointerException.java

public class TryNullPointerException{

sttsic String name;//will intialiazied to null

psvm(String... a){

String name2=null;

sout(name.length());

sout(name2.length());

}

}

TryArithematicException.java

public class TryArithematicException{

psvm(String... a){

int a=0;

int b=10;

int c = b/a;

sout(c);

}

}

TryIllegalArgumentException.java

public class TryIllegalArgumentException{

int numberOfEggs;//this will be intia;lized to zero coz it an intance variable

public void setNumberofEggs(int numberEggs){

if(numberEggs>=0)

this.numberEggs=numberEggs;

else

throw new illegalArgumentException("Number of eggs should be positive");

}

psvm(String... a){

new TryIllegalArgumentException().setNumberOfEggs(-2);

}

}

TryNumberFormatException.java

public class TryNumberFormatException{

psvm(String... a){

sout(Integer.parseInt("abc"));

}

}

Errors

TryExceptionInIntializerError.java

public class TryExceptionInIntializerError{

static{

int arr1[]={1,3,5}

sout(arr1[20]);

}

psvm(String... a){

}

}

TryStackOverFlowException.java

public class TryStackOverFlowException{

psvm(String... a){

doNotCodethis(1);

}

public sttsic void doNotCodethis(int num){

doNotCodethis(1);

}

}

Remember

O Object

T Throwable

E E Exception Error

R Runtime Exception

Approaches to Exception in java

1. handle : wrapped in a compatible try-catch block

Option 1

try{//curly brackets are required

//protected code

}catch(exception\_type indentifier){

//Exception handler

}

Option 2

try(resources){//curly brackets are required

//protected code

}catch(exception\_type indentifier){

//Exception handler

}

Option 3

try{

}catch(exception\_type indentifier){

}finally{

}

e.g

void fall(int distance){

try{

if(distance>10)

throw new IOException();

}catch(IOException ex){

ex.printStackTrace();

}

}

2. declare : state in the method signature

void fall(int distance) throws IOException{

if(distance>10)

throw new IOException();

}

throw vs throws

throw

-an exception is actually being thrown

-this forces the compile to get out of the try/method block

e.g 1

try{

throw new IOException();

System.out.println("Exception");//will not compile coz unreachable statement

}catch(ArrayIndexOutOfBoundsException e){

System.out.println("index of an array out of bound");

}

e.g 2

public static void m1(){

throw new Exception();

System.out.println("Exception");

}

throws

-the method merely has the potential to throw an Exception

We can't use more than one throw statement in one tryblock

void fall(int distance){

try{

if(distance>10){

throw new IOException();

throw new RuntimeException();//Compile error

}

}catch(IOException ex){

ex.printStackTrace();

}

}

Summary - types of exception

type

-Checked exception

-Runtine exception

-Error

how to recognize

-SubClass of RuntimeException but not subclass of RuntimeException

-Subclass of RuntimeException

-Subclass of Error

oky for program to catch

-yes

-Yes

-No

is program required to handle or declare

-yes

-No

-No (this consider bad programming to get Errors)

Remember

-curly brackets are required for try-catch and finally block

try

new throw Exception();

catch(Exception ex)

sout("msg");

//will not compile coz there is no brackets

-The order of the try ststement is try followed by catch then finally block

try{

new throw Exception();

}finally{

sout("msg");

}catch(Exception ex){

sout("msg");

}

//will not compile coz the order should be try-catch-finally

-it followed either catch or finally block or both

try{

throw new Exception();

}

//will not compile coz the try-catch-block is not complete

try-catch-finally block

Option 1

try{

new throw Exception();

}catch(Exception ex){

sout("msg");

}

Option 2

try{

new throw Exception();

}finally{

sout("msg");

}

Option 3

try{

new throw Exception();

}catch(Exception ex){

sout("msg");

}finally{

sout("msg");

}

-When checked exception declared inside a catch block makesure try block is capable of throwing the exception or subclass of the exception

This rule doen't apply to

1. Unchecked exception

2.Exceptions declared in a method signature

3. Exception class

Main.java

class NoMoreCarrotsException extends Exception{

}

public class Main{

psvm(String... a){

try{

eatCarrot();//this method is not throwing an exception will not compile

}catch(NoMoreCarrotsException e){

sout("msg");

}

try{

eatCarrot();//compiles

}catch(NullPointerException e){

sout("msg");

}

try{

good();//compiles

}catch(NoMoreCarrotsException e){

sout("msg");

}

}

public void good throws NoMoreCarrotsException(){

}

private static void eatCarrot(){}

}

Chaning catch-block

-when unchecked exception and its subclasses declared inside catch blocks make sure

- subsclass catch block must be followed by superclass catch block

-can not use exception type outside the catch block

ChanningCatch.java

for uncheckedException

class AnimalsOutForMalk extends RuntimeException{}

class ExhibitClosed extends RuntimeException{}

class ExhibitClosedForLunch extends ExhibitClosed{}

public class ChanningCatch{

psvm(){

try{

seeAnimals();

}catch(AnimalsOutForMalk e){//subclass exception for RuntimeException

sout("try back later");

}catch(ExhibitClosed e){//subclass exception for RuntimeException

sout("not today");

}

}

public void visitMonkey{

//Subclass followed by superclass will compile

try{

seeAnimals();

}catch(ExhibitClosedForLunch e){//subclass exception forExhibitClosed subclass RuntimeException

sout("try back later");

}catch(ExhibitClosed e){//superclass exception for ExhibitClosed

sout("not today");

}

//superclass followed by subclass will not compile

try{

seeAnimals();

}catch(ExhibitClosed e){//superclass exception

sout("not today");

}catch(ExhibitClosedForLunch e){//subclass exception RuntimeException

sout("try back later");

}

}

Unchecked exception RuntimeException

public void visitManatess{

try{

}catch(NumberFormatException e1){

}catch(IllegalArgumentException e2){

sout(e1);//wil not compile coz variable e1 is from the another catch block or e1 is out of scope

}

}

private void seeAnnimal(){

throw new ExhibitClosed();

}

}

Declaring & Override Methods with Exception

-when a class override a method it's not allowed to add new checked exceptions to the method signature

Bunny.java

class CanNotException extends Exception{}

class Mapper{

@Override

public void hop(){

}

@Override

public void hop2(){

}

}

public class Bunny extends Mapper{

@Override

public void hop throws CanNotHopException{//this will not ompile coz the override method hop() was not throwing exception occurs for checked exception

}

@Override

public void hop2 throws illegalArgumentException{//this works for unchecked exception

}

}

An overriden method in the subclass is allowed to declare fewer exceptions than the superclass or interface

Bunny2.java

class CanNotException1 extends Exception{}

class Mapper1{

@Override

public void hop() throws CanNotHopException1{

}

@Override

public void hop2(){

}

}

public class Bunny2 extends Mapper1{

public void hop{//this compiles coz the overriden method can declare few exception than declared on the method in superclass

}

}

-class is allowed to declare a subclass of an exception type

Bunny2.java

class CanNotException2 extends Exception{}

class Mapper2{

public void hop() throws Exception{

}

}

public class Bunny2 extends Mapper2{

@Override

public void hop throws CanNotHopException{//this compile coz the overiden method in superclass throws supertype exception to the one in subclass Bunny2

}

}

-methods are free to throw any unchecked exceptions they want without mentioning them in the method declaration

Parent.java

public class Parent{

protected static Number method(String name) throws Exception{

sout("Parent Method");

return 1;

}

}

class Child extends Parent{

@Override //use @Overriden with OneStatic methods

public final static Integer metod(String name) throws IOException{

//checked exception (same/subtype/No)

//RuntimeException(freely write any where)

//Return type of the overrided method can have (Same/SubType)

//Access modifer of the overrided method can have (Same/Broader)

//Method Signature(Method name & parameters) of the overrided method can have (Same)

//Final method cannot be annotated with @Overriden

//Same Non ststic () / Static (hidding)

sout("third method");

return 2;

}

}

Multi-catch-block

-Exceptions inside multi-catch block aren't related

-Exception can be listed in any order but variable names aprear once

try{

//Protected Code

}catch(Exception1 | Exception2 e){

//single identifier for all exception types

// | is required btwn exception types

//Catch Either of those exceptions

//Exception Handler

}

MultiCatchException.java

public class MultiCatchException{

psvm(String [] args){

try{

sout(args[1]);

}catch(ArrayIndexOutOfBoundsException e){

//e.printStackTrace();

sout("Invalid index to the array");

}catch(NumberFormatException e){

//e.printStackTrace();

sout("Missingf or invalid input");

}

}

}

output

Invalid index to the array

try{

System.out.println(args[1]);

}catch(ArrayIndexOutOfBoundsException | NumberFormatException e){

//e.printStackTrace();

System.out.println("invalid input");

}

SuperType/subClass exception cannot be multi-catched

try{

throw new IOException();

}catch(FileNotFoundException | IOException e){

//e.printStackTrace();

System.out.println("i/o error");

}

Solution

-you just take one type either subclass or super type exception

try{

throw new IOException();

}catch(IOException e){

//e.printStackTrace();

System.out.println("i/o error");

}

Output

i/o error

Multi-catch with different Exception but every Exception in Catch() should be thrown

int a=6;

int b=0;

try{

int d=a/b;

System.out.println(d);

System.out.println(args[0]);

throw new IOException();

}catch(ArithmeticException | IOException | ArrayIndexOutOfBoundsException e){

//e.printStackTrace();

System.out.println("i/o error & arrithematic error & index of an array out of bound");

}

finally Block

-finnally block always excute, weather or not an exception is thrown or a return ststement is used , except encounter System.exit(0);

-finally block whether executes or not an exception occurs or not

public static void main (String[] args) {

try{

m1();//thgis cause StackOverflowError

}catch(StackOverflowError e){

System.out.println("catch-block");

}finally{

System.out.println("finally");

}

}

public static void m1(){

m1();//an error occurs the compiler will not excute the next line

System.out.println("Exception");

}

Output

catch-block

finally

public static void main (String[] args) {

System.out.println(new Main().goHome());

}

int goHome(){

try{

fall();

System.out.println("try ");

return -1;

}catch(Exception e){

System.out.println("catch ");

return -2;

}finally{

System.out.println("finally ");

return -3;

}

}

private void fall() throws IOException{

throw new IOException();

}

output

catch

finally

-3

If there is no Exception to catch , Catch-block will not be excuted

public static void main (String[] args) {

System.out.println(new Main().goHome());

}

int goHome(){

try{

System.out.println("try ");

return -1;

}catch(Exception e){

System.out.println("catch ");

return -2;

}finally{

System.out.println("finally ");

return -3;

}

}

Output:

try

finally

-3

Having two exception on diferent points in the try-catch-finally block

public static void main (String[] args) {

System.out.println(new Main().goHome());

}

String goHome(){

try{

throw new IOException();//primary exception i.e IOException

}finally{

String s=null;

System.out.println("try-finally");

System.out.println(s.length());//supressed exception i.e NullPointerException

return "return finally";//will not be excuted

}

}

Output:

try-finally

if there is primary and supressed exception on a try-catch-block the suppressed exception will be shown

Exception in the catch-block

public static void main (String[] args) {

System.out.println(new Main().goHome());

}

String goHome(){

try{

throw new IOException();//primary exception i.e IOException

}catch(IOException e){

String s=null;

System.out.println(s.length());//this NullPointerException will not be executed to just go to finallyblock

System.out.println("try-catch");

System.out.println("IOException in try");

}finally{

return "return finally";

}

}

Output:

return finally

summary -try statement

legal vs illegal configurations with a traditional try sstement

table>

0 catch blocks

1 or more catch blocks

-> 0 finaly block

Not legal i.e try{}

legal i.e try{}catch(){}

-> 1 finaly block

legal i.e try{}fanally{}

legal i.e try{}catch(){}finally{}

-> 2 or more finaly block

Not legal i.e try{}finally{}finally{}

Not legal i.e try{}catch(){}finally{}finally{}finnaly{}

Automatic resource Management

Resources

-a file/database that requires somekind of stream or connection to read or write data

Resource Leak

-when a program fails to release its connections to the resources, resulting in the resource becoming inaccessible

try-with-resources ststement

-it require resources that implements the AutoCloseable or Closeable interface which includes a void close() method

-it can include multiple resources which are closed after the try clause ends and before any catch/finally block in the reverse order in chich they are declared.

-finally-block in a try-with resource is optional

TryResource.java

public class TryResource{

psvm(string.. a){

}

public void readFile(String file){

FileInputStream = null;

try{

is= new FileInputStream("file-path");

//code to read from the file

}catch(IOException e){

e.printStackTrace();

}finally{

if(is!=null){

try{

is.close();

}catch(IOException e2){

e2.printStackTrace();

}

}

}

}

}

We might forget to close resources but from java 7 we can close resource automatically in reverse order

try(

Connection con=DriverManager.getConnection(url,username,password);

var stmt=con.prepareStatement(sql);

){//this try will close the resources automatically

//and the last semi colon is optional

//do your database operation here

}catch(SQLException ex){

ex.printStackTrace();

}

Implemanting AutoCloseable

MyFileClass.java

public class MyFileClass implements AutoCloseable{

private final int num;//final variable need to be intialized on the line of declaration but it will be intialized on the contrustor

public MyFileClass(int num){

this.num=num;

}

@Override

public void close() throws Exception{

sout("Closing "+num);

}

}

Main.java

public class Main{

psvm(String... a){

try(

MyFileClass f1=new MyFileClass(1);

MyFileClass f1=new MyFileClass(2);

){//this will close in reverse order

throw new RuntimeException();

}catch(Exception e){

sout("expetion in try");

}

}

}

Output

Closing 2

Closing 1

expetion in try

behind the scence it will implicitly create a finally-block but we can add another explicitly finally-block and will compiles

psvm(String... a){

try(

MyFileClass f1=new MyFileClass(1);

MyFileClass f1=new MyFileClass(2);

){

throw new RuntimeException();

}catch(Exception e){

sout("expetion in try");

}finally{

sout("Now in finally-block");

}

}

Output:

Closing 2

Closing 1

expetion in try

Now in finally-block

Autocloseable vs Closeable interface

Closeable extends IOException

Closeable is the super inteface of AutoCloseable

-calling close() method more than once does not have any side effects

-Closeable is an older interface

AutoCloseable extends Exception

AutoCloseable is the subinterface of Closeable

-calling close() method more than once have any side effects

-AutoCloseable is an new interface introduced in java 7

public class MyResource implements AutoCloseable {

public void close() throws Exception {

System.out.println("Closing!");

}

}

try (MyResource res = new MyResource()) {

// use resource here

}

Summary -try-with-resorces statement

-Each resource much include the data type and be separeted by a semicolon(;)

-var can be use as the data type

-try-with-resources is guaranteed s a call the close() method on resources

-catch and explicit finally block are optional

-the resources creted in the try block are in scope only within the try-block

legal vs illegal configurations with a traditional try sstement

table>

0 catch blocks

1 or more catch blocks

-> 0 finaly block

legal i.e try{}

legal i.e try{}catch(){}

-> 1 finaly block

legal i.e try{}fanally{}

legal i.e try{}catch(){}finally{}

-> 2 or more finaly block

Not legal i.e try{}finally{}finally{}

Not legal i.e try{}catch(){}finally{}finally{}finaly{}

public class Main{

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

try(

MyFileClass f1=new MyFileClass(1);

MyFileClass f2=new MyFileClass(2);

){//this will close in reverse order

//throw new RuntimeException();

}

}

}

class MyFileClass implements AutoCloseable{

private final int num;

public MyFileClass(int num){

this.num=num;

}

@Override

public void close() throws Exception{

System.out.println("Closing "+num);

}

}

Scope of try

public void scope() throws Exception{

try(

Scanner s=new Scanner(System.in)

){

s.nextLine();

}catch(Exception ex){

s.nextLine();//compile error

}finally{

s.nextLine();//compile error s is out of scope coz the resource has being close

}

}

Features

Implicitly finally block

-a hiden finally block created by a compiler

-JVM calls the close at the end of the try statement

-it will be called before optional catch and explicity finally block

JammedTurkeyCage.java

public class JammedTurkeyCage implements AutoCloseable{

@Override

public void close throws illegalArgumentException{

throw new illegalArgumentException("Cage door does not close");//SUPPRESSED EXCEPTION

}

psvm(String... a){

try(var t= new JammedTurkeyCage()){

throw new RunetimeException("Turkey ran off");//PRIMARY EXCEPTION

}catch(IllegalArgumentException ex){

sout("catch"+e.getMessage());

}

}

}

-Explicit finally block-declared by the programmer

JammedTurkeyCage.java

public class JammedTurkeyCage implements AutoCloseable{

@Override

public void close throws illegalArgumentException{

throw new illegalArgumentException("Cage door does not close");//SUPPRESSED EXCEPTION

}

psvm(String... a){

try(var t= new JammedTurkeyCage()){

throw new RuntimeException("Turkey ran off");//PRIMARY EXCEPTION

}finally{

throw new RuntimeException("failed to catch it");//SUPPRESSED EXCEPTION

}

}

}

-SUPPRESSED EXCEPTION will be shown

-Effective final

-Suppresed exceptions

-primary exceptions

Create and use Custom exception

Declaring Exception Classes

-when create your exception it is common to extend

- Exception for checked or

-RuntimeException for unchecked

Dophin.java

class CannotSwimException extends Exception{//checked exception

public CannotSwimException(){

super();

}

public CannotSwimException(Exception e){

super(e);

}

public CannotSwimException(String message){

super(message);

}

}

class DangerInException extends RuntimeException{}//unchecked exception

class SharkinTheWaterException extends DangerInException{}

public class Dolhin{

public void swim() throws CannotSwimException{

}

}

Main.java

psvm(String... a) throws CannotSwimException{

//throw new CannotSwimException();

//throw new CannotSwimException("Broken motor");

throw new CannotSwimException(new FileNotFoundException("cannot find shark"));

}

Database Applications with JBBC

-connect to and perform database SQL operations, process queries results using JDBC API

Data

-is the information

Fact

-piece of data e.g first name

Database

-organised collection of data

(a) Relational

-organised into tables have rows and columns

Table

primary key

-unique entry of a particular type in a column

Compound Key

-combination of column

Accessibility

1. Java databse Connectivity (JDBC)

-row and column

2. Java Persistence API(JPA) (not covered by the exam)

- Object RelationMapping (ORM)-java objects

(b) NoSQL (Not in the exam's scope)

-key/values, document stores, graph-based

Creating a Derby Database

Apache Derby

-small in memory datatabase

-for the exam will be using Apache Derby

https://db.apache.org/derby/derby\_downloads.html

mirors.sonic.net/apache/db/derby

SetupDerbyDatabse.java

https://www.selikoff.net/2019/12/20/ocp-11-book-bonus-creating-a-derby-database-in-java-11/

SetupDerbyDatabse.java

public class SetupDerbyDatabase {

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

String url = "jdbc:derby:zoo;create=true";

try (Connection conn = DriverManager.getConnection(url)) {

// run(conn,"DROP TABLE names");

// run(conn,"DROP TABLE exhibits");

run(conn,"CREATE TABLE exhibits ("

+ "id INTEGER PRIMARY KEY, "

+ "name VARCHAR(255), "

+ "num\_acres DECIMAL(4,1))");

run(conn,"CREATE TABLE names ("

+ "id INTEGER PRIMARY KEY, "

+ "species\_id integer REFERENCES exhibits (id), "

+ "name VARCHAR(255))");

run(conn,"INSERT INTO exhibits VALUES (1, 'African Elephant', 7.5)");

run(conn,"INSERT INTO exhibits VALUES (2, 'Zebra', 1.2)");

run(conn,"INSERT INTO names VALUES (1, 1, 'Elsa')");

run(conn,"INSERT INTO names VALUES (2, 2, 'Zelda')");

run(conn,"INSERT INTO names VALUES (3, 1, 'Ester')");

run(conn,"INSERT INTO names VALUES (4, 1, 'Eddie')");

run(conn,"INSERT INTO names VALUES (5, 2, 'Zoe')");

printCount(conn,"SELECT count(\*) FROM names");

}

}

private static void run(Connection conn, String sql) throws SQLException {

try (PreparedStatement ps = conn.prepareStatement(sql)) {

ps.executeUpdate();

}

}

private static void printCount(Connection conn, String sql) throws SQLException {

try (PreparedStatement ps = conn.prepareStatement(sql)) {

ResultSet rs = ps.executeQuery();

rs.next();

System.out.println(rs.getInt(1));

}

}

Visualize the database in Intelij

>Click Database on the right side

>Click the plus Icon

>Select DataPath

Structured Query Language (SQL)

-language to interact with records

Database Definition Language (DDL)

- CREATE

-ALTER

-DROP

Database Manipulation Language (DML)

CRUD operations

operation SQL keywords Description

Create INSERT Adds a new row to the table

Read SELECT Retrieves data from the table

Update UPDATE Changes zero or more rows in the table

Delete DELETE Removes zero or more rows from the table

SELECT count(\*), sum(col\_name) from tbname;

//return row-count & sum items in the columnname

UPDATE tb\_name SET col\_name=col\_name+ .5 WHERE col\_name='kudzie';

Interface of JDBC

-package java.sql

Driver

- estabishes a connection to the database

Connection

-Sends commands to the databse

PreparedStatement

-execute a SQL query

CallableStatement

-Execute commands stored in the databse

ResultSet

-reads results for the query

Interfaces in the JDK

Driver

Connection

PreparedStatement CallableStatement

ResultSet

Implementation in the driver

FooDriver

FooConnection

FooPreparedStatement FooCallableStatement

FooResultSet

For the exam

you should know Driver, Connection, PreparedStatement, ResultSet

Using vanilla JDBC

Managing build tools using Maven(used to manage the dependencies and plugins)

-Download XAMPP

-mysql workbench

poxml.xml

-where you configure dependencies

dependencies

<dependencies>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>88.0.21</version>//version can be optional

</dependency>

</dependencies>

-JDK only provide interfaces (represents contracts & abstraction in java) not implementation of the connection

why JDK provides mainly inteferces rather than implementation classes

-third part libraries they should be independent to Java

-this will allows developers to write one code for many technology

Relationship btwn JPA & JDBC

JPA used hibernate then hibernate uses the JDBC

Connecting to the database

3things need to connect

-Where is the database i.e url

-Creditional i.e username & password

Example1.java

public class Example1{

psvm(String... a){

String url="jdbc:mysql://localhost:3306/db\_name";//localhost means mymachine

String username="root";

String password="";//no space if there is no password

//SQLException is a checked exception

try{

Connection con=DriverManager.getConnection(url,username,password);//this will show a checked Exception that is an expression you need to catch by using try & Catch or otherswise you can throws it on the method

}catch(SQLEXCEPTION e){

e.printStackTrace();

}

}

}

-we don't deal with the connection directly since the connection are managed by the datasource

-Connection

-is the resource so after using it you should close it

-so you should put the resoure in the try() parenthsis so that java will lose the resource

Connection con=null;

try(con= DriverManager.getConnection(url,username,password);

Statement stmt= con.createStatement();

){

//execute() executeUpdate() executeQuery()

stmt.executeUpdate("INSERT INTO product VALUES(null, 'bread',5)");//send a SQL query to the DBMS

}catch(SQLException e){

e.printStackTrace();

}

DataSource

-objects that are implemented to smothen the connection to the database

Specifying the java version your are using in Maven

pom.xml

<build>

<plugins>

<plugin>

<groupId></groupId>

<artfactId></artfactId>

<configuration>

<source>java-version-number</source>

<target>java-version-number</target>

</configuration>

</plugin>

</plugins>

</build>

-Every time you do changes to the pom.xml just reload the mavern

Creating the Statement

- statement is the resources

-most question comes in OCP asks for Statements then PreparedStament often comes

PreparedStatement has a better benefits than Statement

Mysql Methods

execute()

-can be used to execute any SQL query i.e (INSERT, UPDATE, ALTER, CRETE, DROP, TRUNCATE)

-return a boolen i.e true if it was a SELECT and false otherwise

-is the most powerfull method

-used for adminstrative queries i.e CREATE, DROP, TRUNCATE

executeUpdate()

-used to execute INSERT, UPDATE, DELETE SQL query

returns an int i.e the number of rows altered or affected

executeQuery()

-can be used with SELECT

-returns the ResultSet

ResultSet

-parse the Resultset

-get values from specific rows and columns where the cursor is

-Cursor used to select the row at the start the cursor will be above the first row

psvm(){

String url="jdbc:mysql://localhost:3306/db\_name";

String username="";

String password="";

String productName="";

int productPrice=90;

String psql="INSER INTO tb\_name VALUES(null, ? ,?)";

try(var con=DriverManager.getConnection(url,username,password);

//Statement stmt=con.createStatement();

var pstmt=con.prepareStatement(psql);

){

//String sql="INSERT INTO product VALUES(null, 'Flour', 56)";

//Inserting dynamic values

//String sql="INSERT INTO product VALUES(null, '"+productName+"'+","+price+")";

pstmt.setString(1,productName);

pstmt.setDouble(2,price);

pstmt.executeUpdate();

}catch(SQLException ex){

ex.printStackTrace();

}

}

Implementing text block in writting dynmic query

String sql="""

INSERT INTO product VALUES(null, 'Flour', 56)

""";

Example2

Example2.java

public class Example2{

psvm(String... a){

String url="jdbc:mysql://localhost:3306/db\_name";

String username="root";

String password="";

String sql="SELECT \* FROM product";

try{

var con=DriverManager.getConnection(url,username,password);

var pstmt=con.prepareStatement(sql);

){

ResultSet rs=pstmt.executeQuery();

while(rs.next()){//returns true if the row was selected and false if no row was selected

int id=rs.getInt(1);

String name=rs.getString("name");

double price=rs.getDouble("price");

sout(id+ " "+ name+ " "+price);

}

}catch(SQLEXCEPTION e){

e.printStackTrace();

}

}

}

NB

-Only use the query that can have privilategs that the application needs only

-Don't provides many previllages to the application so that it will be easy to secure the application and also easier to work with the application i.e do not give the application more power i.e using the passord of the database superuser

-if the query is dynamaic when the application don't the type of the sql use execute that can execute any query

-if what you need is working with data only use executeQuery() & executeUpdate();

Example 3

dependencies

mysql-connector-java

psvm(){

String url="jdbc:mysql://localhost:3306/db\_name";

String username="";

String password="";

Double price = 4;

String psql="SELECT \* FROM product WHERE price > ?";

try(var con=DriverManager.getConnection(url,username,password);

var pstmt=con.prepareStatement(psql);

){

pstmt.setDouble(1, price);

//ResultSet rs=pstmt.executeQuery();

var rs=pstmt.executeQuery();

while(rs.next()){

int id= rs.getInt("id");//use column label for the application to be mainatainable

String name=rs.getString("name");

double price=rs.getDouble("price");//use BigDecimal coz double does not take of the precision (decimal)

sout(id+" "+ name+ " " + price);

}

}catch(SQLException ex){

ex.printStackTrace();

}

}

DELETING records

String sql="DELETE FROM product WHERE name=?";

{

stmt.setString(1, "product-name");

//int n= stmt.executeUpdate();

boolean result= stmt.execute();

sout(n);//3

sout(result);//false

}

GETTING RESULTSE from .execute()

boolean results=stmt.execute();

ResultSet rs=stmt.getResultSet();

NB

-in real world application don't use var everywhere

-use where the declaration is long

-do not use where we want to see the data type

-if you use the default port there is no need to set it in the url of the database

-in real word senarior use the dataSource rather than DriverManager.getConnection()

-datasource boost the performanace

-Always use custom exceptions in realworld applications

-Try to use Lambok to limit the code fr getters & setters

DataSource

-object that manages multiple connection i.e it reserve connections and save those connection

-we use pooled datasource

-DriverManager is only connecting using url,username,password

JDBC Transactions

-transaction are mannually configured

-jdbc transations not part of ocp 11

tables

account(id,amount)(int,BigDecimal)

dependencies

mysql-connector-java

packages

main

Main.java

public class Main{

psvm(String... args){

MoneyTransferService moneyTransferService= new MoneyTransferService();

moneyTransferService.transferMoney(1,2,new BigDecimal(500));

}

}

model

Account.java

public class Acount{

private int id;

private BigDecimal amount;

//Constructors

//getters & setters

}

repository package

AccountRespository.java

public class AccountRespository{

public Optional<Account> findAccound(Connection con,int id){

String sql="SELECT \* FROM account WHERE id=?";

try(

var stmt=con.prepareStatement(sql);

){

stmnt.setInt(1,id);

var rs=stmt.executeQuery();

if(rs.next()){

var amount=rs.getBigDecimal("amount");

var account=new Account();

account.setId(id);

account.setAmout();

return Optional.of(account);

}else{return Optional.empty();}

}catch(SQL Exception ex){

//throw new DBConnectionFailedException();

ex.printStackTrace();

}

}

public void updateAccount(Connection con,Account account){

String sql="UPDATE account SET amount =? WHERE id=?";

try(

var stmt=con.prepareStatement(sql);

){

stmnt.setBigDecimal(1,account.getAmmount());

stmnt.setInt(2,account.getId());

stmt.executeUpdate();

}catch(SQL Exception ex){

//throw new DBConnectionFailedException();

ex.printStackTrace();

}

}

}

public class AccountRespository{

}

service package

MoneyTransferService.java

public class MoneyTransferService{

private final AccountRepository accountRepository;

/\*public MoneyTransferService(AccountRepository accountRepository){

this.accountRepository=accountRepository;

}\*/

//inshort

public MoneyTransferService(){

this.accountRepository=new AccountRepository();

}

public void transferMoney(int fromAcount,int toAccount,BigDecimal amount){

try(

var con=DBConnectionManager.getInstance().getConnection()

){

var from =accountRepository.findAccount(con,fromAccount);

var to =accountRepository.findAccount(con,toAccount);

var a1=from.orElseThrow(AccountMissingException::new);

var a1=to.orElseThrow(AccountMissingException::new);

a1.setAmount(a1.getAmount().substract(amount));

a2.setAmount(a2.getAmount().add(amount));

accountRepository.updateAccount(con,a1);

if(true) throw new RuntimeException("transaaction");

accountRepository.updateAccount(con,a2);

con.commit();

}catch(SQLException ex){

ex.printStackTrace();

}

}

}

db.manager package

DBConnectionManager.java as SingleTon

public class DBConnectionManager{

//make the constructor private so that can be accesed from the outside

private DBConnectionManager(){}

//creating SingleTon can be used in the many thread

private ststic final class SingletonHolder{

prrivate ststic final DBConnectionManager SINGLETON =new DBConnectionManager();

}

public ststic DBConnectionManager getInstance(){

return SingletonHolder.SINGLETON;

}

pubic Connection getConnection(){

String url="jdbc:mysql://localhost:3306/db\_name";

String username="root";

String password="";

try{

var con=DriverManager.getConnection(url,username,password);

con.setAutoCommit(false);//telling the compiler that it can not commit transaction on itsown but the developer should decide when to commit the transactions

}catch(SQL Exception ex){

throw new DBConnectionFailedException();

}

}

}

>exception

DBConnectionFailedException.java

public class DBConnectionFailedException{

}

AccountMissingException.java

public class AccountMissingException extends RuntimeException{

}

ACID

Atomicity

Consistency

Isolation

Durability