JAVA ASSIGNMENT

Group Members

Avishkaar Pawar AD-1224

Ashish AD-1221

Ankesh

Sanjay

Deepak

**Inheritance**

Q1: How can we access private member and methods in inheritance tree?

Ans: we can use getter methods to get instance variables and we can use java.lang.reflect.\*;

Q2. What if you have same method name in parent and child class which version will be called if you make child class object.

Ans – It will follow bottom to top approach the most specific version which is in child class is called. Method overriding takes place here.

Q3. What if you declare final method in parent class can you override this method in child class?

Ans – No, A final method cannot be overridden. As we intend to not make any change when we use final keyword

Q4. Let us assume three classes A, B, C in which B extends A, C extends B Now we make a reference of B class then ,can we access the methods of A and C class assuming that all methods are public ?

Ans – For A class we can directly access if there is no method overriding else we can use super function for class C. We can not access any method of C class as father cannot inherit from child class

Q5. What will happen if we make a class both final and abstract ?

Ans – As abstract keyword are used to declare at a later state and final is used to make something immutable both are contrary things, this is like we want to go in two opposite direction in same time, it will give compile time error.

Q6. How can we access sibling classes ?

Ans. We must make classes and methods as public or at least default access . then we can access class like this MyClass.method() in sibling class.

**Interfaces**

Q1.What will happen if you implement an interface in a class, and forgets to declare the abstract method body from interface in class? Why this error occurs?

Ans- Abstract methods need to be declared somewhere at last and make a concrete definition. Otherwise it causes error as no formal body is defined anywhere for that method in any class .

Q2. Class and interfaces both have extension “.class” when compiled? So any guesses why we needed a new keyword “ interface”?

Ans-We can’t inherit from two classes in java , as java supports only single inheritance.There occurs a deadly situation in multiple inheritance called “Deadly Diamond of death” which occurs if same thing exist in two parents, compiler sees it as ambiguous choice. So to overcome this we uses interfaces ,where interfaces have abstract method to deal it.

Q3. How extends and different keyword works differently for classes and interfaces?

Ans- In classes only one class can be inherited , i.e. class MyClass extends Interface1{} , but in interfaces we can extend s class from multiple interface , i.e. interface MyInterface extends Interface1, Interface2 {} .

Q4. Are nested interfaces possible ?If yes , then how can we access a sub-interface?

Ans- Yes , nested interfaces are possible . to access static methods , where “Intf” is interface name and it contains nested interface named “Inner” . We can access it like , Intf.Inner.method();

Q5. If you want to store a lot of constants . how can you achieve it with interfaces?

Ans- By default all variables in interface are static, final and public. Static and final acts as constants and can be used without creating objects and are unchangeable.

Q6. Can we create non static variables in interfaces?

Ans-No.We can not create non static variables in an interface. If we try to create non static variables compile time error comes.

**Packages**

Q1.Is it necessary to put a package statement ?What happens if we forget to put it? Where they will be stored and in which package?

Ans- No , there isn’t any compulsion to put a package statement. If we don’t put a package command then it goes in default package.

Q2. Does importing a package make sub-packages class files available to the application?

Ans- The import statement imports only classes of the package, but not sub-packages and its classes.

Q3. What happens in background when you run “javac -d . \*.name” ?

Ans- Java compiler select the destination folder and “ .” selects current directry and makes folder structure automatically if any package is there.

Q4. Can we import same package twice?

Ans- Yes , it will not cause error as the class is already loaded once and checked by JVM by Class ID before another same class import.

Q5.What is benefit of using specific class name , rather than full package?

Ans-If we import full packages , unnecessary codes also comes into scope , in large application there is a large probability that we can have ambiguity in names of variables or method , as same name can exist in two different packsges that we are importing.

Q6. How does Java knows a package exist and where to find it?

Ans-  "$JAVA\_HOME/jre/lib/ext” packages are checked here .

 –classpath or -cp command-line option can be used to set user defined classes .

**Exception Handling**

Q1.Give Practical case to differentiate exception and error.

Ans- Error are unchecked , occurs at runtime. Suppose we running a program and memory runs out, then we see OutOfMemoryError .

Exception are checked , can occur at rutime or compile time. Suppose we are using a pointer to refer a object and later on we that objects is delted, then we gets a exception NullPointerException .

Q2. What will happen if you forget to handle an error?

Ans- When an exception occurred, if you don’t handle it, the program terminates abruptly and the code past the line that caused the exception will not get executed.

Q3.What if an error is thrown before a try block is completed?

Ans- The code after that throw line is unreachable and causes error: unreachable statement

Q4.  **Can we write only try block without catch and finally blocks?**

**Ans-** No, It shows compilation error. The try block must be followed by either catch or finally block. You can remove either catch block or finally block but not both.

Q5. Throw vs throws vs Throwable?

**throw** is a keyword in java which is used to throw an exception manually

**throws** is also a keyword in java which is used in the method signature to indicate that this method may throw mentioned exceptions

**Throwable** is a super class for all types of errors and exceptions in java

Q6. Can we throw primitive type exceptions?

Ans- No , only class that belongs to Exception class and subclass can be thrown.

**PRACTICAL QUESTIONS**

**Inheritance**

1) class A{

void get(){

System.out.println("This is class A.");

}

}

class B {

void get(){

System.out.println("This is class B.");

}

}

class C extends A,B{

void get(){

System.out.println("This is class C.");

}

}

class Run{

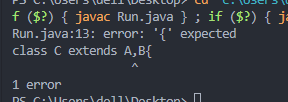
public static void main(String args[]){

C obj = new C();

obj.get();

}

}



We cannot extend multiple classes. Only one class need to be there.

2)

class SuperClass{

protected String name;

}

class SubClass extends SuperClass {

private String name;

public static void setName(String name) {

super.name = name;

}

public void display() {

System.out.println("name: "+super.name);

}

}

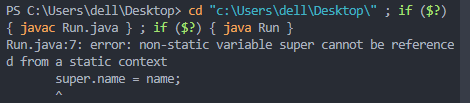
class Run{

public static void main(String args[]) {

new SubClass().display();

}

}



In a static method we want to update a non static variable . This causes error.

3)

class A{

void get(){

System.out.println("This is class A.");

}

}

class B {

void get(){

System.out.println("This is class B.");

}

}

class C extends A,B{

void get(){

System.out.println("This is class C.");

}

}

class Run{

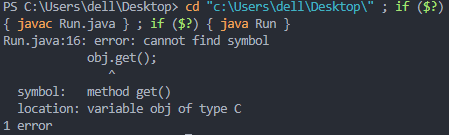
public static void main(String args[]){

C obj = new C();

obj.get();

}

}



We are trying to access a private method from inheriting class.

4)

class Shape{

void draw(){

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

}

}

class Rectangle extends Shape{

void draw(){

System.out.println("drawing rectangle...");

}

}

class Run{

public static void main(String args[]){

Shape a = new Shape();

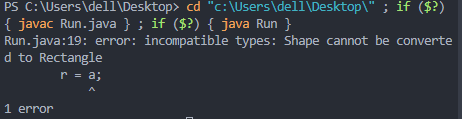
Rectangle b = new Rectangle();

Rectangle r;

r = a;

r.draw();

}

}

Incomaptible type conversion .We are trying to hold a parent class object in a child class reference.

5)

class A{

abstract void get();

}

class B extends A{

void let(){

System.out.println("This is class B.");

}

}

class Run{

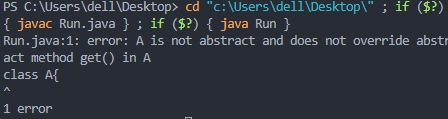
public static void main(String args[]){

C obj = new C();

obj.get();

}

}



Method get() isn’t declared anywhere in concrete form.

6)

public class Run {

public static void main(String [] args){

}

}

class Animal{

Animal(){

System.out.println("Animal Created");

}

}

class Dog{

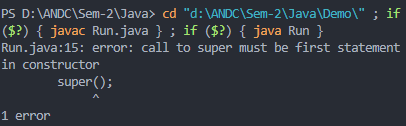
Dog(){

System.out.println("Dog Created");

super();

}

}



Super must be the first statement in constructors.

**Packages**

1)

class A1 {

public void msg(){

System.out.println("This is inside class A");

}

}

package P;

public class Run{

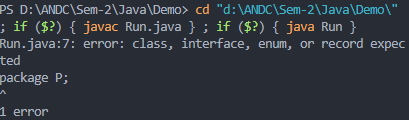
public static void main(String args[]){

A test = new A();

test.msg();

}

}



We get error because package must be the first line that a compiler must read .( Ignoring comments)

2)

package p;

public class A {

private int x=8;

}

package p1;

import p.A;

public class Run{

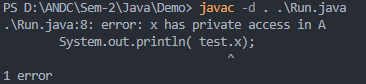
public static void main(String args[]){

A test = new A();

System.out.println( test.x);

}

}



We are trying to access variable that isn’t visible out of package.

3)

package Run;

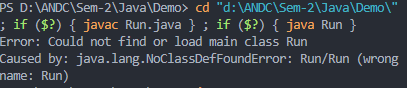
public class Run{

public static void main(String [] args){

System.out.println("Hulla");

}

}



This error occurs in most IDE , as default code-runners don’t run “java Run” instead of “java Run.Run”

4)

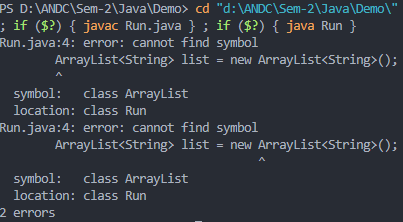
public class Run {

public static void main(String [] args){

ArrayList<String> list = new ArrayList<String>();

}

}



ArrayList is a class in java.util which is needed to be imported first.

5)

package p;

public class Run {

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

}

Suppose we have a previous Run.class file in package ‘p’ , it will be rewrited and causes data loss . To avois such cases we must use uniqye names.

6)

import p;

public class Run {

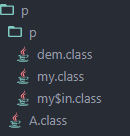
public static void main(String [] args){

dem obj = new dem();

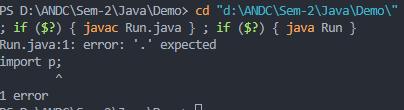
dem.show();

}

}



Here we need to declare atleast one more level of hierarchy like import p.\*;



**Interface**

1)

public class Ok implements my{

public static void main(String [] args){

Ok obj=new Ok();

obj.show();

}

}

interface my implements asdf,zxcv{

int x=10;

default void show(){

System.out.println(x+" "+y+" "+z);

}

}

interface asdf{

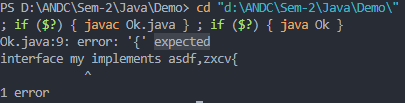
int y=19;

}

interface zxcv{

int z=20;

}



If we replace implements with extends , the code works.

2)

public class Ok implements my{

public static void main(String [] args){

Ok obj=new Ok();

obj.show();

}

}

interface my{

int x=10;

default void show(){

System.out.println("My interface");

}

}

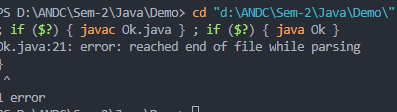
class my{

int x=10;

void show(){

System.out.println("My class");

}



We cannot name a class and interface with same name in a file.

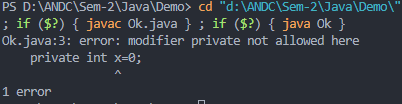
3)

interface A {

private int x=0;

void meth1( );

}



Modifier protected and private don’t work here , s by default these values are public.

4)

interface A {

void m1();

interface In {

void m2();

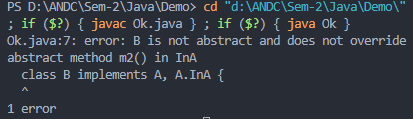
}

}

class B implements A, A.InA {

public void m1(){

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



Abstract method m2() is not being made concrete ever here.

5) interface A {

abstract void m1();

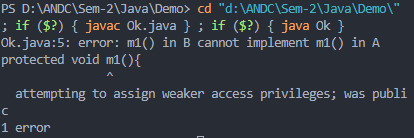
}

class B implements A {

protected void m1(){

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

}



We are explicitly trying to reduce access level of method ,which causes error.

6)

interface A {

abstract void m1();

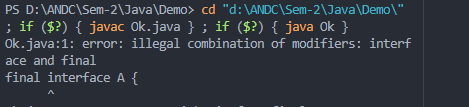
}

class B implements A {

protected void m1(){

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

}



We can’t use final and interface together.

**Exception Handling**

1)

class ExceptionHandling{

public static void main(String args[]){

try{

int i = Integer.parseInt("XYZ");

}catch(Exception e){

System.out.println("This block handles all exception types");

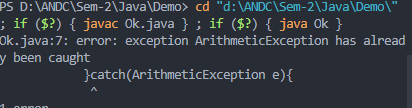
}catch(ArithmeticException e){

System.out.println("This block will give an error");

}

}

}



Order in which we declare catch block changes everything , if superclass Exception is defined first then , the later catch will be unreachable and thus unnecessary.

2)

class ExceptionHandling{

public static void main(String args[]){

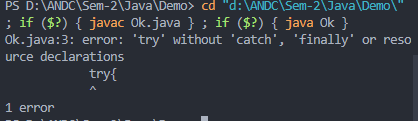
try{

double i= 100/0.0;

}System.out.println("This line will generate an error. ");

}

}



Alteast one thing out of catch or finally is needed after try block

3)

class ExceptionHandling{

public static void main(String args[]){

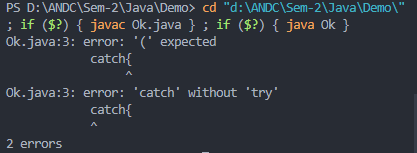
catch{

double i= 100/0.0;

}System.out.println("This line will generate an error. ");

}

}



catch without try causes error .

4) class x{

void method(){

try{

double i= 100/0.0;

}

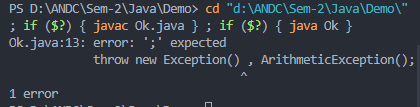
catch(Exception e){

throw new Exception() , ArithmeticException();

}

}

}



We can’t throw two Exceptions at same time.

5)

class x{

void method() throws new Exception(){

try{

double i= 100/0.0;

}

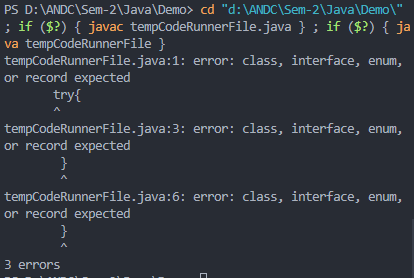
catch(Exception e){

throw new Exception() ;

}

}

}



Because we need to pass class , not instance of any exception class.

6)

package UnderAge;

import java.util.Scanner;

class UnderAge {

int age;

UnderAge(int age){

this.age=age;

}

public String toString(){

String temp="Under Age: "+age;

return temp;

}

}

public class UnderAgeDemo{

static void test(int age){

try {

if (age<18){

throw new UnderAge(age);

}

else{

System.out.println("age is above 18");

}

}

catch (UnderAge a){

System.out.println(a.toString());

}

}

public static void main(String []args ){

Scanner sc=new Scanner(System.in);

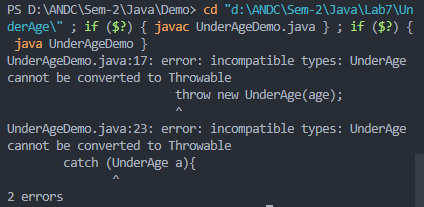
System.out.println("Enter the age: ");

int age=sc.nextInt();

test(age);

}

}



Our custom exception class must extend Throwable or its sub class like Exception.