1.Why do we need to handle the exception? And how do we handle the exception? Explain with example

Ans:

Exception is an error event that can happen during the execution of a program and disrupts its normal flow.Exceptions in java can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure, Database server down etc.

Java being an object oriented programming language, whenever an error occurs while executing a statement, creates an exceptionobject and then the normal flow of the program halts and JRE tries to find someone that can handle the raised exception.

The exception object contains a lot of debugging information such as method hierarchy, line number where the exception occurred, type of exception etc. When the exception occurs in a method, the process of creating the exception object and handing it over to runtime environment is called **“**throwingtheexception**”**.

Following are some scenarios where an exception occurs.

* A user has entered an invalid data.
* A file that needs to be opened cannot be found.
* A network connection has been lost in the middle of communications or the JVM has run out of memory.

2. What is an Exception? Explain about kinds of Exceptions

Ans.

Exception is an error event that can happen during the execution of a program and disrupts its normal flow.

Java’s exceptions can be categorized into two types:

* Checked exceptions
* Unchecked exceptions
* Error

Errors:

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Checked Exceptions:

Checked exceptions − A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.

|  |  |
| --- | --- |
| **Name** | **Description** |
| IOException | While using file input/output stream related exception |
| SQLException. | While executing queries on database related to SQL syntax |
| DataAccessException | Exception related to accessing data/database |
| ClassNotFoundException | Thrown when the JVM can’t find a class it needs, because of a command-line error, a classpath issue, or a missing .class file |
| InstantiationException | Attempt to create an object of an abstract class or interface. |

Unchecked Exceptions:

Unchecked exceptions − An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

|  |  |
| --- | --- |
| **Name** | **Description** |
| NullPointerException | Thrown when attempting to access an object with a reference variable whose current value is null |
| ArrayIndexOutOfBound | Thrown when attempting to access an array with an invalid index value (either negative or beyond the length of the array) |
| IllegalArgumentException. | Thrown when a method receives an argument formatted differently than the method expects. |
| IllegalStateException | Thrown when the state of the environment doesn’t match the operation being attempted,e.g., using a Scanner that’s been closed. |
| NumberFormatException | Thrown when a method that converts a String to a number receives a String that it cannot convert. |
| ArithmeticException | Arithmetic error, such as divide-by-zero. |

Unchecked Exceptions:

1. **int** a=50/0;//ArithmeticException
2. String s=**null**; System.out.println(s.length());//NullPointerException
3. String s="abc";

**int** i=Integer.parseInt(s);//NumberFormatException

1. **int** a[]=**new** **int**[5];

a[10]=50; //ArrayIndexOutOfBoundsException

3. Differentiate between an Error and Exception

Ans

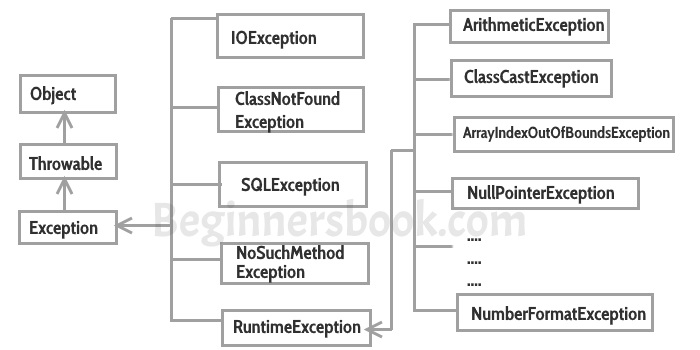
Errors − These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

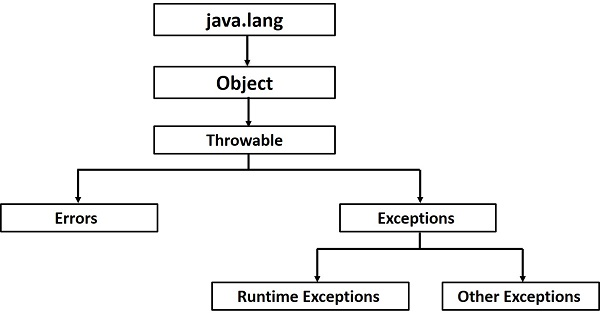
Exception is an error event that can happen during the execution of a program and disrupts its normal flow.Exceptions in java can arise from different kind of situations

such as wrong data entered by user, hardware failure, network connection failure, Database server down etc.

4. Explain about Exception Hierarchy

Ans





5. Differentiate between throw and throws with examples

Ans

1. Throwsclause is used to declare an exception, which means it works similar to the try-catch block. On the other hand throw keyword is used to throw an exception explicitly.

2. If we see syntax wise than throw is followed by an instance of Exception class and throws is followed by exception class names.

throw new ArithmeticException(“Arithmetic exception”);

throws ArithmeticException(“arithmetic exception”);

3. Throw keyword is used in the method body to throw an exception, while throws is used in method signature to declare the exceptions that can occur in the statements present in the method.

4. You can throw one exception at a time but you can handle multiple exceptions by declaring them using throws keyword.

7.

import java.util.Scanner;

class Division {

public static void main(String[] args) {

int a, b, result;

Scanner input = new Scanner(System.in);

System.out.println("Input two integers");

a = input.nextInt();

b = input.nextInt();

result = a / b;

System.out.println("Result = " + result);

}

}

Output:

Input two integers

2

0

Stops execution

(b)

class Division {

public static void main(String[] args) {

int a, b, result;

Scanner input = new Scanner(System.in);

System.out.println("Input two integers");

a = input.nextInt();

b = input.nextInt();

try {

result = a / b;

System.out.println("Result = " + result);

}

catch (ArithmeticException e) {

System.out.println("Exception caught: Division by zero.");

}

}

}

Output:

Exception caught: Division by zero

(c)

class Exceptions {

public static void main(String[] args) {

String languages[] = { "C", "C++", "Java", "Perl", "Python" };

try {

for (int c = 1; c <= 5; c++) {

System.out.println(languages[c]);

}

}

catch (Exception e) {

System.out.println(e);

}

}

}

Output:

C++

Java

Perl

Python

java.lang.ArrayIndexOutOfBoundsException: 5

8. What is Serialization and DeSerialization in java? explain with examples

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

The ObjectOutputStream class contains writeObject**()** method for serializing an Object.The ObjectInputStream class contains readObject() method for deserializing an object.

**Points to remember:**

1. If a parent class has implemented Serializable interface then child class doesn’t need to implement it but vice-versa is not true.

2. Only non-static data members are saved via Serialization process.

3. Static data members and transient data members are not saved via Serialization process.So, if you don’t want to save value of a non-static data member then make it transient.

4. Constructor of object is never called when an object is deserialized.

5. Associated objects must be implementing Serializable interface.

package extended\_java\_prgs;

import java.io.\*;

import java.io.Serializable;

public class Serialization\_Deserialization\_Example {

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

DemoSerialization ds=new DemoSerialization(101,"praveen");

try{

FileOutputStream fo=new FileOutputStream("/home/sb-33/Desktop/Output");

ObjectOutputStream oos=new ObjectOutputStream(fo);

oos.writeObject(ds);

System.out.println("Object has been Serialized");

fo.close();

oos.close();

}

catch(IOException e) {

System.out.println("Exception caught:"+e);

}

try{

FileInputStream fis=new FileInputStream("/home/sb-33/Desktop/Output");

ObjectInputStream ois=new ObjectInputStream(fis);

DemoSerialization obj=null;

obj=(DemoSerialization)ois.readObject();

System.out.println("id:"+obj.i);

System.out.println("name:"+obj.n);

System.out.println("Object has been Deserialized");

fis.close();

ois.close();

}

catch(IOException e) {

System.out.println(e);

}

}

}

class DemoSerialization implements Serializable{

int i;

String n;

DemoSerialization(int id,String name) {

this.i=id;

this.n=name;

}

}

9. What is Byte Stream in java?

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, FileInputStream and FileOutputStream. Following is an example which makes use of these two classes to copy an input file into an output file −

public class SimpleFilePrograms {

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

FileInputStream in=null;

FileOutputStream out=null;

in=new FileInputStream("/home/sb-33/Desktop/Sample");

out=new FileOutputStream("/home/sb-33/Desktop/Output");

int c;

while((c=in.read())!=-1) {

System.out.print((char)c);

out.write(c);

}

in.close();

out.close();

}

}

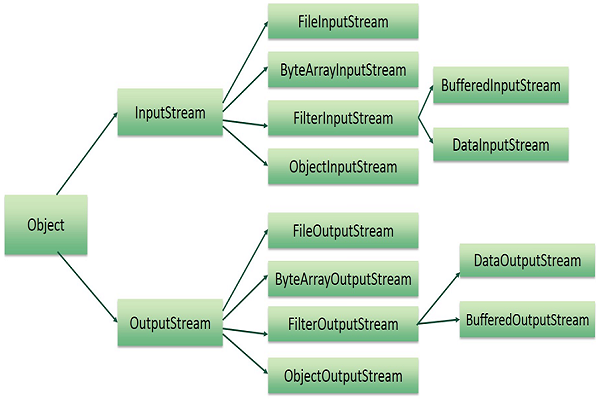
Character Stream:

JavaByte streams are used to perform input and output of 8-bit bytes, whereas Java Character streams are used to perform input and output for 16-bit Unicode. Though there are many classes related to character streams but the most frequently used classes are,FileReader and FileWriter**.** Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

A byte stream is suitable for processing raw data like binary files.

Character stream is useful when we want to process text files.

Names of character streams typically end with Reader/Writer and names of byte streams end with InputStream/OutputStream

10. What is InputStream and What is OutputStream

Java performs I/O through Streams. a stream means continuous flow of data.

Java encapsulates Stream under java.io package. Java defines two types of streams. They are,

1. Byte Stream : It provides a convenient means for handling input and output of byte.
2. Character Stream : It provides a convenient means for handling input and output of characters. Character stream uses Unicode and therefore can be internationalized

A stream can be defined as a sequence of data. There are two kinds of Streams −

* InPutStream − The InputStream is used to read data from a source.
* OutPutStream − The OutputStream is used for writing data to a destination.

11. Mention about the methods used in FileInputStream and FileOutputStream

public class SimpleFileProgram {

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

FileInputStream in=null;

FileOutputStream out=null;

in=new FileInputStream("/home/sb-33/Desktop/Sample");

out=new FileOutputStream("/home/sb-33/Desktop/Output");

int c;

while((c=in.read())!=-1) {

System.out.print((char)c);

out.write(c);

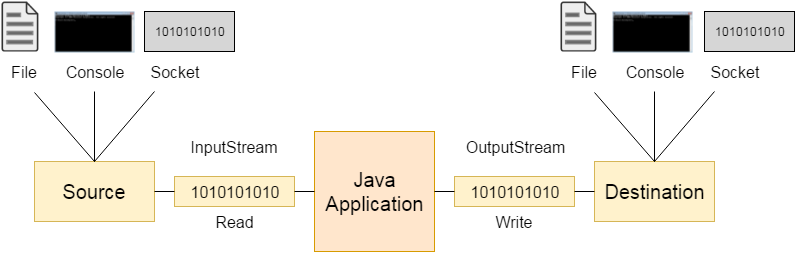
}

in.close();

out.close();

}

}



12. Write a program for following

a. create a directory

b. In that directory add a file called text.txt

c. add content "This is a demo text file" in text.txt

d. Read the text.txt file

Ans:

package oops;

import java.io.\*;

public class CreateDirectoryFileAddReadContent {

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

String s="/home/sb-33/Desktop/sam1";

//creating Directory

File f=new File(s);

if(f.mkdir())

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

else

System.out.println("Failed to create Directory");

//Rename Directory

File old=new File("/home/sb-33/Desktop/sam1");

File newDir=new File("/home/sb-33/Desktop/sample");

if(old.renameTo(newDir))

System.out.println("renamed");

else

System.out.println("failed to rename");

//Creating file inside Directory

File fileObj=new File("/home/sb-33/Desktop/sample/text.txt");

if(fileObj.createNewFile())

System.out.println("File Created inside Directory");

else

System.out.println("Failed to create a file");

/\* //Adding content using CharacterStream object

FileWriter writer=new FileWriter(fileObj);

String content="This is a Demo text file";

writer.write(content);

writer.close();

\*/

//adding Content using ByteStream object

FileOutputStream fo=new FileOutputStream(fileObj);

String content="This is a demo text file";

fo.write(content.getBytes());

fo.close();

}

}

13. Program to check if a file or directory physically exist or not.

package oops;

import java.io.\*;

public class CheckFileOrDirectoryExistsOrNot {

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

File f=new File("/home/sb-33/Desktop/Output");

FileInputStream fin=new FileInputStream(f);

if(f.exists()) {

System.out.println("Readable:"+f.canRead());

System.out.println("Writeable:"+f.canWrite());

System.out.println("Directory:"+f.isDirectory());

System.out.println("size"+f.length());

}

System.out.println("Name:"+f.getName());

System.out.println("path:"+f.getPath());

System.out.println("AbsolutePath:"+f.getAbsolutePath());

System.out.println("Exists:"+f.exists());

System.out.println("parent:"+f.getParent());

}

}

14 Consider a file contains a paragraph of data. Write a program to read the file line by line

Ans

package oops;

import java.io.\*;

public class ReadDataLinebyLineFromFile {

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

FileReader fr=new FileReader("/home/sb-33/Desktop/Output");

BufferedReader br=new BufferedReader(fr);

BufferedWriter br1=new BufferedWriter(new FileWriter("/home/sb-33/Desktop/Input"));

byte c=0;

String str;

while((str=br.readLine())!=null) {

System.out.println(str);

br1.write(str);

br1.write("\n");

c++; //Counting number of lines

}

System.out.println("Lines:"+c);

br1.close();

}

}

17. What is garbage collection?

* In C/C++, programmer is responsible for both creation and destruction of objects. Usually programmer neglects destruction of useless objects. Due to this negligence, at certain point, for creation of new objects, sufficient memory may not be available and entire program will terminate abnormally causing OutOfMemoryErrors.
* But in Java, the programmer need not to care for all those objects which are no longer in use. Garbage collector destroys these objects.
* Garbage collector is best example of Daemon thread as it is always running in background.
* Main objective of Garbage Collector is to free heap memory by destroying unreachable objects.

Daemon thread is a low priority thread that runs in background to perform tasks such as garbage collection.

Unreachable objects :An object is said to be unreachable iff it doesn’t contain any reference to it. Also note that objects which are part of island of isolation are also unreachable

Object 1 references Object 2 and Object 2 references Object 1. Neither Object 1 nor Object 2 is referenced by any other object. That’s an island of isolation.

There are generally four different ways to make an object eligible for garbage collection.

* 1. Nullifying the reference variable
  2. Re-assigning the reference variable
  3. Object created inside method
  4. Island of Isolation

18. Write a program to perform garbage collection

Ans

package oops;

public class GarbageCollectorExample {

public static void main(String args[]) {

GarbageCollectorExample g1=new GarbageCollectorExample();

GarbageCollectorExample g2=new GarbageCollectorExample();

// can call Runtime.getRuntime().gc();

System.gc();

g1=null;

g2=null;

//g1.close();

//g2.close();

}

//finalize method is called before colleting objects

public void finalize() throws Throwable {

System.out.println("Garbage Collector called");

}

}

Just before destroying an object, Garbage Collector calls *finalize()* method on the object to perform cleanup activities. Once *finalize()* method completes, Garbage Collector destroys that object.

*finalize()* method is present in Object class with following prototype.

Protected void finalize() throws Throwable

19. Explain about the Ways for requesting JVM to run Garbage Collector with example

We can also request JVM to run Garbage Collector. There are two ways to do it :

* 1. Using *System.gc()* method : System class contain static method *gc()* for requesting JVM to run Garbage Collector.
  2. Using *Runtime.getRuntime().gc()* method : Runtime class allows the application to interface with the JVM in which the application is running. Hence by using its gc() method, we can request JVM to run Garbage Collector.

public class GarbageCollectorExample {

public static void main(String args[]) {

GarbageCollectorExample g1=new GarbageCollectorExample();

GarbageCollectorExample g2=new GarbageCollectorExample();

// can call Runtime.getRuntime().gc();

System.gc();

g1=null;

g2=null;

//g1.close();

//g2.close();

}

//finalize method is called before colleting objects

public void finalize() throws Throwable {

System.out.println("Garbage Collector called");

}

}

20. What is Enumeration? Explain with an example

Enumerations serve the purpose of representing a group of named constants in a programming language. For example the 4 suits in a deck of playing cards may be 4 enumerators named Club, Diamond, Heart, and Spade, belonging to an enumerated type named Suit. Other examples include natural enumerated types (like the planets, days of the week, colors, directions, etc.).

Enums are represented using enum data type. Java enums are more powerful than C/C++ enums . In Java, we can also add variables, methods and constructors to it. The main objective of enum is to define our own data types(Enumerated Data Types).

Enum declaration can be done outside a Class or inside a Class but not inside a Method.

First line inside enum should be list of constants and then other things like methods, variables and constructor.

Every enum constant is always implicitly publicstaticfinal. Since it is static, we can access it by using enum Name. Since it is final, we can’t create child enums.

Every enum constant represents an object of type enum.

enum can contain concrete methods only i.e. no any abstract method.

package extended\_java\_prgs;

public class Enum\_Example {

public static void main(String args[]) {

Day day=Day.MONDAY;

System.out.println("Day:"+ day.TUESDAY);

day.demoMethod();

}

enum Day {

MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY,SUNDAY;

public void demoMethod() {

System.out.println("From Enum");

}

}

}

21. Consider a variable Directions of enum type, which is a collection of four constants EAST, WEST, NORTH and SOUTH.

Create a class EnumDemo and make use of enum variable. (example if you get the enum value is EAST. you should display "you are at EAST direction")

Ans

package extended\_java\_prgs;

public class EnumDemo\_Directions {

enum Directions {

EAST, WEST, NORTH, SOUTH;

}

public static void main(String args[]) {

Directions dir=Directions.WEST;

if(dir==Directions.NORTH) {

System.out.println("You are at NORTH Directions");

}

else if(dir==Directions.WEST) {

System.out.println("You are at WEST Directions");

}

else if(dir==Directions.EAST) {

System.out.println("You are at EAST Directions");

}

else if(dir==Directions.SOUTH) {

System.out.println("You are at SOUTH Directions");

}

}

}

22. Explain about Autoboxing and Unboxing with an example

Ans

**Autoboxing:** Converting a primitive value into an object of the corresponding wrapper class is called autoboxing. For example, converting int to Integer class. The Java compiler applies autoboxing when a primitive value is:

* Passed as a parameter to a method that expects an object of the corresponding wrapper class.
* Assigned to a variable of the corresponding wrapper class.

**Unboxing:** Converting an object of a wrapper type to its corresponding primitive value is called unboxing. For example conversion of Integer class to int. The Java compiler applies unboxing when an object of a wrapper class is:

* Passed as a parameter to a method that expects a value of the corresponding primitive type.
* Assigned to a variable of the corresponding primitive type.

package extended\_java\_prgs;

public class AutoBoxing\_UnboxingExample {

public static void main(String args[]) {

Integer i=new Integer(10);

Character ch=new Character('p');

//autoBoxing

int a=i;

char c=ch;

System.out.println("autoboxing:"+a+c);

//Unboxing

Integer unInt=a;

Character unChar=c;

System.out.println("unboxing:"+unInt+unChar);

}

}

23. What are annotations in java? Explain with an example

Annotations are used to provide supplement information about a program.

* Annotations start with ‘**@**’.
* Annotations do not change action of a compiled program.
* Annotations help to associate *metadata* (information) to the program elements i.e. instance variables, constructors, methods, classes, etc.
* Annotations are not pure comments as they can change the way a program is treated by compiler. See below code for example.

Java defines seven built-in annotations.

* Four are imported from java.lang.annotation: **@**Retention, **@**Documented, **@**Target, and **@**Inherited.
* Three are included in java.lang: **@**Deprecated**, @**Overrideand **@**SuppressWarnings