**What is the advantage of using exception handling?**

exception handling enables a program to deal with exceptional situations andcontinue its normal execution.Runtime errors occur while a program is running if the JVM detects an operation that is  
impossible to carry out. For example, if you access an array using an index that is out of  
bounds, you will get a runtime error with an Array Index Out Of Bounds Exception**.** If you  
enter a double value when your program expects an integer, you will get a runtime error with  
an Input Mismatch Exception.  
runtime errors are thrown as exceptions. An exception is an object that represents  
an error or a condition that prevents execution from proceeding normally. If the exception is  
not handled, the program will terminate abnormally.

==================================================================

* **Which of the following statements will throw an exception?**

System. out println(**1** / **0**);

Exception kind :Arithmetic Exception: / by zero

=====================================================================

* **Point out the problem in the following code. Does the code throw any exceptions?**

**long** value = Long. MAX\_VALUE + **1**;

System . out. println(value);

=== max value is 263-1 value override

Does not throw exception

==============================================

* ***What does the JVM do when an exception occurs? How do you catch an exception?***

The JVM spins itself up and prepares the execution environment. The JVM creates a thread which will run the main() method using whatever command-line parameters are applicable. The JVM sets **a default uncaught exception handler** that prints the exception to standard error and terminates. The JVM executes the thread

* What is the output of the following code?

***try{int value =30;***

***if(value < 40)***

***throw new Exception("value is too small");}catch (Exception ex) {***

***System. out. println(ex. get Message());***

***}***

***System. out. println("continue after catch block");***

**Output :**

value is too small

continue after catch block

**Show the output of the following code**

**public class test1 {**

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

**Scanner in = new Scanner(System.in);**

**for (int i = 0; i < 2; i++) {**

**System.out.print(i+" ");**

**try{**

**System.out.println(1/0);**

**}catch (Exception ex) {**

**} } }}**

**OutPut:**

**0,1**

**public class test1 {**

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

**Scanner in = new Scanner(System.in);**

**try{**

**for (int i = 0; i < 2; i++) {**

**System.out.print(i+" ");**

**System.out.println(1/0);**

**}}catch (Exception ex) {**

**} }}**

**OutPut:**

**0**

* **What is a checked exception, and what is an unchecked exception?**

**checked exceptions: are forced by the compiler and used to indicate exceptional conditions that are out of the control of the program.**

**unchecked exceptions:  are occurred during runtime and used to indicate programming errors**

* **Suppose that causes an exception in the following try-catch block: Answer the following questions:**
  + **Will be executed?======🡺NO**
  + **If the exception is not caught, will   
    be executed?====================🡺Yes**
  + **If the exception is caught in the catch block,   
    will be executed?======🡺YES**
* **Correct a compile error in the following code:**

**public void m(int value) {**

**if (value < 40)**

**try {**

**throw new Exception("value is too small");**

**} catch (Exception ex) {**

**Logger.getLogger(test1.class.getName()).log(Level.SEVERE, null, ex);**

**}**

}

========================================================================

**Summarize the methods of 𝐷𝑎𝑡𝑎𝑔𝑟𝑎𝑚𝑆𝑜𝑐𝑘𝑒𝑡 and 𝐷𝑎𝑡𝑎𝑔𝑟𝑎𝑚𝑃𝑎𝑐𝑘𝑒𝑡 classes**

**Datagram socket method**

|  |  |
| --- | --- |
| [**bind**](https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html#bind(java.net.SocketAddress))(**[SocketAddress](https://docs.oracle.com/javase/7/docs/api/java/net/SocketAddress.html" \o "class in java.net)** addr) | Binds this DatagramSocket to a specific address & port. |
| [**connect**](https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html#connect(java.net.SocketAddress))(**[SocketAddress](https://docs.oracle.com/javase/7/docs/api/java/net/SocketAddress.html" \o "class in java.net)** addr) | Connects this socket to a remote socket address (IP address + port number) |
| [**getLocalSocketAddress**](https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html#getLocalSocketAddress())() | Returns the address of the endpoint this socket is bound to. |
| [**getReceiveBufferSize**](https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html#getReceiveBufferSize())() | Get value of the SO\_RCVBUF option for this DatagramSocket, that is the buffer size used by the platform for input on this DatagramSocket |
| [**receive**](https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html#receive(java.net.DatagramPacket))(**[DatagramPacket](https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html" \o "class in java.net)** p) | Receives a datagram packet from this socket |
| [**send**](https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html#send(java.net.DatagramPacket))(**[DatagramPacket](https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html" \o "class in java.net)** p) | Sends a datagram packet from this socket. |
| [**setReuseAddress**](https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html#setReuseAddress(boolean))(boolean on) | Enable/disable the SO\_REUSEADDR socket option. |
| [**connect**](https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html#connect(java.net.InetAddress,%20int))(**[InetAddress](https://docs.oracle.com/javase/7/docs/api/java/net/InetAddress.html" \o "class in java.net)** address, int port) | Connects the socket to a remote address for this socket. |

**𝐷𝑎𝑡𝑎𝑔𝑟𝑎𝑚𝑃𝑎𝑐𝑘𝑒𝑡 method**

|  |  |
| --- | --- |
| 1) InetAddress getAddress() | It returns the IP address of the machine to which the datagram is being sent or from which the datagram was received. |
| 2) byte[] getData() | It returns the data buffer. |
| 3) int getLength() | It returns the length of the data to be sent or the length of the data received. |
| 4) int getOffset() | It returns the offset of the data to be sent or the offset of the data received. |
| 5) int getPort() | It returns the port number on the remote host to which the datagram is being sent or from which the datagram was received. |
| 6) SocketAddress getSocketAddress() | It gets the SocketAddress (IP address + port number) of the remote host that the packet is being sent to or is coming from. |
| 7) void setAddress(InetAddress iaddr) | It sets the IP address of the machine to which the datagram is being sent. |
| 8) void setData(byte[] buff) | It sets the data buffer for the packet. |
| 9) void setLength(int length) | It sets the length of the packet. |
| 10) void setPort(int iport) | It sets the port number on the remote host to which the datagram is being sent. |
| 11) void setSocketAddress(SocketAddress addr) | It sets the SocketAddress (IP address + port number) of the remote host to which the datagram is being sent. |

**What is 𝑆𝑜𝑐𝑘𝑒𝑡 class?**

This class implements client sockets (also called just "sockets"). A socket is an endpoint for communication between two machines.

The actual work of the socket is performed by an instance of the SocketImpl class. An application, by changing the socket factory that creates the socket implementation, can configure itself to create sockets appropriate to the local firewall

**What is 𝐼𝑛𝑒𝑡𝐴𝑑𝑑𝑟𝑒𝑠𝑠 class is used for?**

The **java.net.InetAddress** class provides methods to get the IP address of any hostname. An IP address is represented by 32-bit or 128-bit unsigned number. InetAddresscan handle both IPv4 and IPv6 addresses

**What are unicast and broadcast?**

1. **Unicast :**  
   Unicast is a type of information transfer and it is used when there is a participation of single sender and single recipient. So, in short, you can term it as a one-to-one mapping. For example, a device having IP address 10.1.4.0 in a network wants to send the traffic stream (data packets) to the device with IP address 20.14.4.2 in the other network, then unicast comes into the picture. It is the most common form of data transfer over the networks.
2. **Broadcast :**  
   Broadcast transfer (one-to-all) techniques and can be classified into two types : Limited Broadcasting, Direct Broadcasting. In broadcasting mode, transmission happens from one host to all the other hosts connected on the LAN. The devices such as bridge uses this. The protocol like ARP implement this, in order to know MAC address for the corresponding IP address of the host machine. ARP does ip address to mac address translation. RARP does the reverse.

**What are the differences between unicast, multicast, and broadcast?**

* **Unicast**: from one source to one destination i.e. One-to-One
* **Broadcast**: from one source to all possible destinations i.e. One-to-All
* **Multicast**: from one source to multiple destinations stating an interest in receiving the traffic i.e. One-to-Many

**What are java Generics and wildcards?**

**Generics:**

**Generics** mean **parameterized types**. The idea is to allow type (Integer, String, … etc, and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types.   
An entity such as class, interface, or method that operates on a parameterized type is called a generic entity.

**Wildcards:**

The question mark (?) is known as the wildcard in generic programming . It represents an unknown type. The wildcard can be used in a variety of situations such as the type of a parameter, field, or local variable; sometimes as a return type. Unlike arrays, different instantiations of a generic type are not compatible with each other, not even explicitly. This incompatibility may be softened by the wildcard if ? is used as an actual type parameter.

**What is the difference between array list and enums?**

**They are completely different data structures with completely different usages.**

**An array** is a group of variables of same type that are referred to by a common name. Array can contains primitives data types as well as objects of a class depending on the definition of array. In case of primitives data types, the actual values are stored in contiguous memory locations. In case of objects of a class, the actual objects are stored in the heap.

**Enum** is a keyword in java and is a type like class or interface and it can be used to define set of enum constants. Enum are collection of named constants, Once you declared Enum constants you cannot change there value . Enum’s are type-safe can be used in switch cases.