1) What is an exception? Explain exception handing mechanism with proper example.

Ans:- An exception is an event that disrupts the normal flow of a program during its execution. It is a way for the program to indicate that something unexpected or erroneous has occurred. When an exception occurs, it can be thrown by the Java Virtual Machine (JVM) or explicitly by the program itself. Exceptions are objects representing an abnormal condition in the program.

Exception handling in Java is done through the use of five keywords: `try`, `catch`, `throw`, `throws`, and `finally`.

1. `try`: The `try` block encloses the code that might throw an exception. It is followed by one or more `catch` blocks and/or a `finally` block.

2. `catch`: The `catch` block is used to handle exceptions. If an exception is thrown inside the `try` block, it is caught by the appropriate `catch` block based on the type of the exception.

3. `throw`: The `throw` keyword is used to explicitly throw an exception. You can throw both built-in exceptions (like `NullPointerException`, `ArithmeticException`, etc.) or custom exceptions (user-defined exceptions).

4. `throws`: The `throws` keyword is used in the method signature to indicate that the method may throw certain types of exceptions. The caller method must handle these exceptions using a `try-catch` block or propagate them using the `throws` keyword.

5. `finally`: The `finally` block contains code that needs to be executed regardless of whether an exception is thrown or not. It is often used for cleanup tasks like closing files or releasing resources.

```
import java.util.Scanner;

public class Example {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a number: ");
        try {
            int number = Integer.parseInt(scanner.nextLine());
            int result = 10 / number;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero.");
        }
    }
}
```

2) How to define i. multiple catch blocks ii. Nested try blocks.

Ans:- i. Multiple Catch Blocks:

In Java, you can have multiple `catch` blocks to handle different types of exceptions. When an exception is thrown, the catch blocks are evaluated in order, and the first one that matches the type of the exception is executed. If no catch block matches the exception type, the program terminates with an error message. Here's an example demonstrating multiple catch blocks:

```
public class MultipleCatchExample {
    public static void main(String[] args) {
        try {
            int[] numbers = {1, 2, 3};
            System.out.println(numbers[5]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index out of bounds error.");
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic error occurred.");
        } catch (Exception e) {
            System.out.println("Some other exception occurred.");
        }
    }
}
```

ii. Nested Try Blocks:

Java also allows you to nest `try` blocks, meaning you can place one `try` block inside another. This is useful when you want to handle exceptions differently in different parts of your code. Here's an example of nested try blocks:

```
public class NestedTryExample {
    public static void main(String[] args) {
        try {
            // Outer try block
            int[] numbers = {1, 2, 3};
            System.out.println(numbers[5]); // This may throw an ArrayIndexOutOfBoundsException

            try {
                // Inner try block
                String text = null;
                System.out.println(text.length()); // This will throw a NullPointerException
            } catch (NullPointerException e) {
                System.out.println("Null pointer exception occurred inside inner try block.");
            }
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index out of bounds error occurred in outer try block.");
        }
    }
}
```

3) Explain throw, throws and finally with example program.

Ans:-

```
import java.util.Scanner;

public class Calculator {
   public static double divide(double numerator, double denominator) throws ArithmeticException {
      if (denominator == 0) {
         throw new ArithmeticException("Cannot divide by zero.");
      }
      return numerator / denominator;
   }
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      double numerator, denominator;
      double result = 0;
      try {
         System.out.print("Enter numerator: ");
         numerator = scanner.nextDouble();
         System.out.print("Enter denominator: ");
         denominator = scanner.nextDouble();
         result = divide(numerator, denominator);
         System.out.println("Result of division: " + result);
      } catch (ArithmeticException e) {
         System.out.println("Error: " + e.getMessage());
      } finally {
         System.out.println("Inside finally block. Closing resources...");
         scanner.close();
      }
   }
}
```

4) Demonstrate the concept of garbage collection through suitable examples.

Ans:- Garbage collection in Java is the process of automatically deallocating memory occupied by objects that are no longer reachable or in use by the program. Java's garbage collector runs in the background, automatically freeing up memory by removing objects that are no longer referenced. Let's go through an example to demonstrate how garbage collection works:

```
class MyClass {

private String name;

public MyClass(String name) {

      this.name = name;

}

public void finalize() {

     System.out.println(name + " object is being garbage collected.");

   }

 }
```

```
public class GarbageCollectionExample {
    public static void main(String[] args) {
        MyClass obj1 = new MyClass("Object 1");
        MyClass obj2 = new MyClass("Object 2");
        obj1 = null;
        obj2 = null;
        System.gc();
    }
}
```

5) Define package. Explain the creation of a package using a suitable example program

Ans:- In Java, a package is a way to organize related classes and interfaces into a single namespace. It helps prevent naming conflicts and provides a modular and structured approach to designing Java applications. Packages are directories that store Java files. By using packages, you can group related classes and interfaces together, making it easier to manage and maintain your code.

Here's how you can create a package and use it in a Java program:

```
import com.example.MyClass;
public class Main {
    public static void main(String[] args) {
        MyClass myObject = new MyClass();
            myObject.display();
    }
}
```

6) Define synchronization? Explain how inter-thread communication can be achieved in multithreading using producer and consumer problem

Ans:- In multithreading, synchronization is the concept of controlling the access of multiple threads to shared resources. When multiple threads access shared data concurrently, there is a possibility of data inconsistency, as one thread might modify the data while another is reading it. Synchronization ensures that only one thread can access the shared resource at a time, preventing data corruption and maintaining data integrity.

In Java, synchronization can be achieved using the **`synchronized`** keyword. Methods or blocks can be marked as synchronized to ensure that only one thread can execute them at a time.

```
class Buffer {
    private int item;
    private boolean isEmpty = true;
    synchronized void produce(int newItem) {
        while (!isEmpty) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        item = newItem;
        isEmpty = false;
        System.out.println("Produced: " + item);
        notify();
    }
        synchronized int consume() {
        while (isEmpty) {
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        isEmpty = true;
        System.out.println("Consumed: " + item);
        notify();
        return item;
    }
}
```

```
class Producer implements Runnable {
    private Buffer buffer;
    Producer(Buffer buffer) {
        this.buffer = buffer;
    }
    public void run() {
        for (int i = 1; i <= 5; i++) {
            buffer.produce(i);
        }}}
class Consumer implements Runnable {
    private Buffer buffer;
    Consumer(Buffer buffer) {
        this.buffer = buffer;
    }
    public void run() {
        for (int i = 1; i <= 5; i++) {
            buffer.consume();
        }}}
public class ProducerConsumerExample {
    public static void main(String[] args) {
        Buffer buffer = new Buffer();
        Producer producer = new Producer(buffer);
        Consumer consumer = new Consumer(buffer);
        Thread producerThread = new Thread(producer);
        Thread consumerThread = new Thread(consumer);
        producerThread.start();
        consumerThread.start();
    }
}
```

7) Define multithreading? Construct a java program to create multiple threads in Java by implementing runnable interface

Ans:- **Multithreading** is the concurrent execution of two or more threads. A thread is a lightweight process, and multithreading is a popular technique to improve the performance and responsiveness of applications. Multithreading allows multiple threads to execute concurrently within the same program. Each thread runs independently, allowing developers to perform multiple tasks simultaneously, making the most out of modern multi-core processors.

```
class MyRunnable implements Runnable {
   private String threadName;
   MyRunnable(String name) {
      this.threadName = name;
   }
   public void run() {
      for (int i = 1; i <= 5; i++) {
         System.out.println("Thread " + threadName + ": Count " + i);
         try {
            // Sleep for a random time between 1 to 1000 milliseconds
            Thread.sleep((long) (Math.random() * 1000));
         } catch (InterruptedException e) {
            System.out.println("Thread " + threadName + " interrupted.");
         }
      }
      System.out.println("Thread " + threadName + " finished.");
   }
}
public class MultiThreadExample {
   public static void main(String[] args) {
      System.out.println("Main thread started.");
      Thread thread1 = new Thread(new MyRunnable("A"));
      Thread thread2 = new Thread(new MyRunnable("B"));
      thread1.start();
      thread2.start();
 System.out.println("Main thread finished.");
   }}
```