I - ASSIGNMENT

(Start Writing From Here)

1. Create a Java program to compute the sum of the digits of a given integer number use BufferedReader for user input. Explain the usage of operator new with an example.

Here's a simple Java program that computes the sum of the digits of a given integer using BufferedReader for user input:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Digitsumcalculator{
    public static void main (String [] args) {
        try {
            // create a BufferedReader Object to read user input
            BufferedReader reader = new BufferedReader (new
                                    InputStreamReader (system.in));
            System.out.print ("Enter an integer number:");
            String input = reader.readLine();
            int number = Integer.parseInt(input);
            //Calculate the sum of digits
            int digitSum = calculateDigitSum (number);
            // Display the result
            System.out.println ("Sum of digits:" +digitSum);
            // Close the Buffered Reader
            reader.close();
```

```
} catch (IOException | NumberFormatException e) {
        System.out.println("Error: " + e.getMessage());
    }
}

// method to calculate the sum of digits in a number
private static int calculateDigitSum(int num) {
        int sum = 0;
        // Loop through each digit and add it to the sum.
        while (num != 0) {
            sum += num % 10; // Get the last digit
            num /= 10; // Remove the last digit
        }
        return sum;
    }
}
```

Explanation of new operator:

The new operator in Java is used to create an instance of a class or to create an array. It dynamically allocates memory for an object or an array and returns a reference to that memory location.

```
// Create a new instance of the String class
String myString = new String("Hello, World!");
// Create a new array of integers
int[] myArray = new int[5];
```

In the provided code, new InputStreamReader(System.in) is using the new operator to create a new instance of InputStreamReader that reads from the standard input stream (System.in). This instance is then used to create a BufferedReader object, allowing us to read user input from the console.

2. Distinguish between widening and narrowing in java with example.

In Java, widening and narrowing refer to type conversions between primitive data types.

Widening (Implicit conversion):

Occurs when a smaller data type is converted into a larger data type. No data loss during conversion. Happens automatically by compiler.

Example:

```
int smallerInt = 10;
long largerLong = smallerInt; // widening conversion from int to long
```

Narrowing (Explicit conversion):

Occurs when a larger datatype is converted to a smaller data type. May result in data loss because the larger datatype might not fit into the smaller one. Requires explicit casting by the programmer.

Example:

```
long largerLong = 1000;
int smallerInt = (int) LargerLong; // Narrowing conversion from long to int with explicit casting.
```

In the narrowing example, the programmer explicitly uses (int) to indicate that they are aware of the potential loss of data when converting a long to an int.

3. Explain the uses of super with an Example. Does a super class reference be used to refer sub class object? Justify.

In object-oriented programming, the super keyword is used to refer to the superclass (parent class) of the current subclass (child class). It is often used to call methods or access fields from the superclass.

Here's an example in Java:

```
class Animal {
    void eat() {
        system.out.println("This animal eats food.");
    }
}

class Dog extends Animal{
    void eat(){
        super.eat(); // calling the eat() method of the
                     super class.
```

```
        System.out.println("The dog eats bones.");
    }
}
public class Main {
    public static void main(String[] args){
        Dog myDog = new Dog();
        myDog.eat();
    }
}
```

output:

This animal eats food.

The dog eats bones.

In this example, the Dog class extends the Animal class. The eat method in the Dog class uses super.eat() to call the eat method of the superclass (Animal), and then it adds additional functionality specific to the Dog class.

A superclass reference can be used to refer to a subclass reference can be used to refer to a subclass object through polymorphism. This allows flexibility and code reusability.

For Example:

```
Animal mypet = new Dog();
mypet.eat(); // calls the overridden eat() method in the
             dog class.
```

Here, mypet is declared as an Animal reference but points to a Dog object. The method called is the one in the Dog class because of polymorphism, demonstrating that a superclass reference can indeed be used to refer to a subclass object.

4. Illustrate Dynamic binding with a suitable example.

Dynamic binding is a concept in programming where the method or function to be executed is determined at runtime, rather than at compile time. In object-oriented languages like Java, dynamic binding is often associated with polymorphism.

Consider a scenario in Java with a base class shape and two subclasses Circle and square. Each subclass has its own implementation of the draw method. Here's a simple example:

```
class shape {
    void draw() {
        System.out.println("Drawing a shape");
    }
}
class Circle extends shape {
    void draw() {
        System.out.println("Drawing a circle");
    }
}
```

```
class square extends shape{
    void draw(){
        system.out.println("Drawing a square");
    }
}

public class main{
    public static void main (String[] args ){
        Shape shape1 = new circle();
        shape shape2 = new square ();
        shape1.draw(); // output: Drawing a circle
        shape2.draw(); // output: Drawing a square
    }
}
```

In this example, shape1 and shape2 are declared as objects of the base class shape, but they are instantiated as objects of the subclasses circle and square respectively. when calling the draw method on these objects, the actual method that gets executed is determined at runtime based on the type of object.

This is dynamic binding in action- the decision on which draw method to invoke is made dynamically during program execution based on the actual type of the object, promoting flexibility and extensibility in the code.

5. Sketch the Exception hierarchy and also give the benefits of Exception Handling. Is it essential to catch all types of exceptions? Justify your answer

The exception hierarchy typically includes a base class, often named Exception, with more specific exception classes inheriting from it. some languages have variations, but the structure generally follows this pattern.

Benefits of Exception Handling:

① Error localization: Helps identify where the error occured in the code, aiding in debugging.

② maintainability: seperates error-handling code from normal code, making it easier to maintain and understand.

③ Graceful Termination: Allows a program to terminate gracefully when unexpected situations arise, preventing abrupt crashes.

④ Customization: Enables developers to create custom exception classes for specific error scenarios.

⑤ Robustness: Enhances program robustness by handling unexpected situations and preventing them from disrupting the normal flow.

It's not always essential to catch all types of exceptions. Catching only specific exceptions relevant to the anticipated errors is often preferable. Catching all exceptions can mask bugs or unexpected issues, making it harder to diagnose problems. Additionally, catching overly board exceptions might lead to less precise error handling.

Justification:

specificity: catching specific exceptions allows for targeted and appropriate responses to different error scenarios.

Clarity: Focusing on relevant exceptions enhances code readability and makes it clear how different errors are handled.

Avoiding silencing Bugs: catching all exceptions might hide programming errors, making it challenging to identify and fix issues.

In summary, while exception handling is crucial, it's advisable to catch only the exceptions that you can reasonably handle and let others propagate for better debugging and maintainability