```java
/////////////// FILE HEADER //////////////////////////
//
// Title: CS 542 - Exercise 3.2: Numeric Errors
// Course: CS 542 Introduction to Software Security, Fall 2022
//
// Author: Binhao Chen, Steven Yang
// Email: bchen276@wisc.edu, yang558@wisc.edu
// Instructors' Name: Barton Miller, Elisa Heymann
//
////////////////////////////////////////////////////////////////////////////////

/**
 * This class includes functions (add, subtract, multiply, and divide) that do
 * 32-bit signed integer arithmetic returning a correct 32-bit signed integer
 * result or throw an exception when overflow occurs.
 *
 * @author Binhao Chen, Steven Yang
 */
public class NumericErrors {

 /**
  * Add two integers and deal with the cases of possible overflow
  *
  * Corner Cases:
  *      When x >= 0 AND y >= 0:
  *    If: x+y < 0     =>    x+y > Integer.MAX_VALUE  =>   overflow occurs
  *   When x <= 0 AND y <= 0:
  *    If: x+y > 0     =>    x+y < Integer.MIN_VALUE  =>   overflow occurs
  *
  * Otherwise, no overflow occurs for this addition operation.
  *
  */
 public static int add(int x, int y) throws ArithmeticException {
  int sum = x + y;
  try {
   // Case 1:
   if (x >= 0 && y >= 0) {
    if (sum < 0) {
     throw new ArithmeticException("int overflow: add(" + x + ", " + y + ")");
    }
   }

   if (x <= 0 && y <= 0) {
    if (sum > 0) {
     throw new ArithmeticException("int overflow: add(" + x + ", " + y + ")");
    }
   }
  } catch (ArithmeticException e) {
   System.out.println(e.getMessage());
   return -1;
  }

  return sum;
 }

 /**
  * Subtract two integers and deal with the cases of possible overflow
  *
  * * Corner Cases:
  *      When x >= 0 AND y <= 0:
  *    If: x-y < 0     =>    x-y > Integer.MAX_VALUE  =>   overflow occurs
  *   When x <= 0 AND y >= 0:
  *    If: x-y > 0     =>    x-y < Integer.MIN_VALUE  =>   overflow occurs
  *
  * Otherwise, no overflow occurs for this addition operation.
  *
  */
 public static int subtract(int x, int y) throws ArithmeticException {
  int diff = x - y;
  try {
   if (x >= 0 && y <= 0) {
    if (diff < 0) {
     throw new ArithmeticException("int overflow: subtract(" + x + ", " + y + ")");
    }
   }

   if (x <= 0 && y >= 0) {
    if (diff > 0) {
     throw new ArithmeticException("int overflow: subtract(" + x + ", " + y + ")");
    }
   }
  } catch (ArithmeticException e) {
   System.out.println(e.getMessage());
   return -1;
  }

  return diff;
 }
```

```java
/**
 * Multiply two integers and deal with the cases of possible overflow
 *
 *  Question: Why is this harder than doing the sum or difference?
 *  Answer: Multiplication will generate a much larger overflow which is harder to detect than by only checking if the result is
 *          negative or positive as in the sum and difference. Multiply two large postive number that cause overflow may still
 *          generate a postive value due to the wrapping.
 *
 *  Corner Cases:
 *   1. x > 0:
 *    a. y > Integer.MAX_VALUE / x     =>    x*y > Integer.MAX_VALUE
 *       b. y < Integer.MIN_VALUE / x     =>    x*y < Integer.MIN_VALUE
 *      2. x < 0:
 *    a. x == -1 && y == Integer.MIN_VALUE    =>  x*y > Integer.MAX_VALUE since  |Integer.MIN_VALUE| = Integer.MAX_VALUE + 1
 *    b. y < Integer.MAX_VALUE / x     =>    x*y > Integer.MAX_VALUE
 *       c. y > Integer.MIN_VALUE / x     =>    x*y < Integer.MIN_VALUE
 */
public static int multiply(int x, int y) throws ArithmeticException {
 try {
  if (x > 0) {
   if (y > Integer.MAX_VALUE / x || y < Integer.MIN_VALUE / x) {
    throw new ArithmeticException("int overflow: multiply(" + x + ", " + y + ")");
   }
  } else if (x < 0) {
   if (x == -1 && y == Integer.MIN_VALUE) {
    throw new ArithmeticException("int overflow: multiply(" + x + ", " + y + ")");
   }

   if (y < Integer.MAX_VALUE / x || y > Integer.MIN_VALUE / x) {
    throw new ArithmeticException("int overflow: multiply(" + x + ", " + y + ")");
   }
  }
 } catch (ArithmeticException e) {
  System.out.println(e.getMessage());
  return -1;
 }
 return x * y;
}

/**
 * Divide two integers and deal with the cases of possible overflow
 *
 *  Question: Is overflow detection necessary?
 *  Answer: Yes. The special case where Integer.MIN_VALUE / (-1) will cause overflow since |Integer.MIN_VALUE| = Integer.MAX_VALUE + 1
 *
 *  Corner Cases:
 *   1. y == 0: Divide by 0 exception will occur
 *   2. x == Integer.MIN_VALUE && y == -1 because |Integer.MIN_VALUE| = Integer.MAX_VALUE + 1
 */
public static int divide(int x, int y) {
 try {
  if (y == 0) {
   throw new ArithmeticException("Divide by zero exception: divide(" + x + ", " + y + ")");
  }

  if (x == Integer.MIN_VALUE && y == -1) {
   throw new ArithmeticException("int overflow: divide(" + x + ", " + y + ")");
  }
 } catch (ArithmeticException e) {
  System.out.println(e.getMessage());
  return -1;
 }

 return x / y;
}

/**
 * Main Function, with several corner tests.
 *
 */
public static void main(String args[]) {

 // Test cases for add()
 // Corner cases:
 System.out.println(add(100, Integer.MAX_VALUE-10));
 System.out.println(add(2, Integer.MAX_VALUE-1));
 System.out.println(add(-100, Integer.MIN_VALUE+10));
 System.out.println(add(-2, Integer.MIN_VALUE+1));

 System.out.println(add(9, Integer.MAX_VALUE-10));
 System.out.println(add(1, Integer.MAX_VALUE-1));
 System.out.println(add(-9, Integer.MIN_VALUE+10));
 System.out.println(add(-1, Integer.MIN_VALUE+1));


 // Test cases for subtract()
 // Corner cases:
```

```
    System.out.println(subtract(Integer.MAX_VALUE-10, -100));
    System.out.println(subtract(Integer.MAX_VALUE-1, -2));
    System.out.println(subtract(Integer.MIN_VALUE+10, 100));
    System.out.println(subtract(Integer.MIN_VALUE+1, 2));

    System.out.println(subtract(Integer.MAX_VALUE-10, -9));
    System.out.println(subtract(Integer.MAX_VALUE-1, 2));
    System.out.println(subtract(Integer.MIN_VALUE+10, 8));
    System.out.println(subtract(Integer.MIN_VALUE+1, -2));


    // Test cases for multiply()
    System.out.println(multiply(5, 429496730));
    System.out.println(multiply(5, -429496730));
    System.out.println(multiply(-1, Integer.MIN_VALUE));
    System.out.println(multiply(-10, -214748365));
    System.out.println(multiply(-10, 214748365));
    System.out.println(multiply(10000, 10000));

    // Test cases for divide()
    System.out.println(divide(5, 0));
    System.out.println(divide(Integer.MIN_VALUE, -1));
    System.out.println(divide(-1, Integer.MIN_VALUE));
  }
}
```