```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.InputMismatchException;
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;

/**
 * @author Warren Smith.
 *
 */
public class Main {

  /**
   * Main Method.
   *
   * @param args
   *          Not used.
   */
  public static void main(String[] args) {

    doTwoDimensionalArrayStuff();

    goToTheStore();

    stringProcessing();
  }

  /**
   * Everything encompassing 2D Arrays.
   */
  private static void doTwoDimensionalArrayStuff() {
    Scanner scan;
    final int sizeX = 3;
    final int sizeY = 3;
    int[][] arr = new int[sizeY][sizeX];

    // Explain to User
    System.out.println(
        "We will now calculate the sum of an hourglass within a 3x3 2D Array: \n");
    System.out.println("Enter 9 numbers (one at a time) between -9 and 9: \n");

    boolean doneWithInput = false;
    do {
      scan = new Scanner(System.in);
      int index = sizeY * sizeX;
      doneWithInput = true;

      // Loop through 2D Array
      for (int i = 0; i < sizeX; i++) {
        for (int j = 0; j < sizeY; j++) {

          // Exception Handling
          try {
            arr[i][j] = scan.nextInt();
            if (arr[i][j] > 9 || arr[i][j] < -9) {
              throw new InputMismatchException();
```

```
58            }
59            System.out.println(arr[i][j] + " ... " + --index + " left to go!");
60          } catch (InputMismatchException e) {
61            System.out.println(
62                "That's not an integer from -9 to 9. Try (all over) again.");
63            // break out of loops
64            i += sizeX;
65            j += sizeY;
66            doneWithInput = false;
67          } catch (Exception e) {
68            System.out.println(e);
69            // break out of loops
70            i += sizeX;
71            j += sizeY;
72            doneWithInput = false;
73          }
74        }
75      }
76    } while (!doneWithInput);
77
78    System.out.println("Here's what you input: \n");
79    for (int i = 0; i < sizeX; i++) {
80      for (int j = 0; j < sizeY; j++) {
81        System.out.print(" " + arr[i][j]);
82      }
83      System.out.println("");
84    }
85
86    System.out.println("\n");
87
88    ArrayList<Integer> possibleValues = new ArrayList<Integer>();
89    for (int i = 0; i < sizeX; i++) {
90      for (int j = 0; j < sizeY; j++) {
91        // If we have room to the right and below then find Hourglass value
92        if (i + 2 < 3 && j + 2 < 3) {
93          possibleValues.add(findValueOfHourglass(i, j, arr));
94        }
95      }
96    }
97
98    // Sum of Hourglass
99    System.out.print("And here's the sum of your hourglass: "
100       + Collections.max(possibleValues));
101
102   // Sleep Thread so User can Read
103   goToSleep(3000);
104
105   // Search a Two-Dimensional Array and Identify the Coordinates Where a Value
106   // was Found!
107   System.out.println("\n\n"
108       + "Now we will find the coordinates of a number within this 2D array! \n");
109   System.out
110       .print("Enter a number that you used for one of your 9 numbers: ");
111
112   int num = 0;
113   // Input Handling
114   do {
```

```
115       scan = new Scanner(System.in);
116       doneWithInput = true;
117       // Exception Handling
118       try {
119         num = scan.nextInt();
120         if (num > 9 || num < -9) {
121           scan.close();
122           throw new InputMismatchException();
123         }
124       } catch (InputMismatchException e) {
125         System.out.println("WTF NO");
126         doneWithInput = false;
127       } catch (Exception e) {
128         System.out.println(e);
129         doneWithInput = false;
130       }
131     } while (!doneWithInput);
132
133     findCoordinatesOfHourglassNumber(arr, num, sizeX, sizeY);
134
135     // close Scanner
136     // scan.close(); -> closing scanner causes infinite loop later
137   }
138
139   /**
140    * Method used to find the total value of the "I" (hourglass).
141    *
142    * @param startingX
143    *          this is the x coordinate passed.
144    * @param startingY
145    *          this is the y coordinate passed.
146    * @param paramValues
147    *          these are the numbers accumulated for the final total.
148    * @return
149    *          returns accumulation of values.
150    */
151   private static int findValueOfHourglass(int startingX, int startingY,
152       int[][] paramValues) {
153     int accumulator = 0;
154     for (int i = startingX; i < startingX + 3; i++) {
155       for (int j = startingY; j < startingY + 3; j++) {
156         if ((i == startingX || i == startingX + 2) && j == startingY + 1) {
157           continue;
158         }
159         accumulator += paramValues[j][i];
160       }
161     }
162     return accumulator;
163   }
164
165   /**
166    * Method to find Coordinates for the User-defined number in the hourglass.
167    *
168    * @param arr
169    *          array containing all numbers of hourglass.
170    * @param numberToFind
171    *          the number to find (user-defined).
```

```
172   * @param sizeX
173   *          size of array in x dimension.
174   * @param sizeY
175   *          size of array in y dimension.
176   */
177  private static void findCoordinatesOfHourglassNumber(int[][] arr,
178      int numberToFind, int sizeX, int sizeY) {
179    LinkedList<Vector2> queue = new LinkedList<Vector2>();
180    for (int i = sizeX - 1; i >= 0; i--) {
181      for (int j = sizeY - 1; j >= 0; j--) {
182        if (numberToFind == arr[j][i]) {
183          queue.push(new Vector2(i, j));
184        }
185      }
186    }
187    System.out.println();
188
189    switch (queue.size()) {
190      case 0:
191        System.out.println("Apparently we couldn't find that number.");
192        break;
193      case 1:
194        System.out.println("We found that number at the following location:");
195        System.out.println("( X, Y )");
196        System.out.println("________");
197        queue.pop().print();
198        break;
199      default:
200        System.out
201            .println("We found a few instances at the following locations:");
202        System.out.println("( X, Y )");
203        System.out.println("________");
204        while (!queue.isEmpty()) {
205          queue.pop().print();
206        }
207        break;
208    }
209  }
210
211  /**
212   * Method to sleep thread (so user can read text).
213   *
214   * @param timeInMilliseconds
215   *          number of milliseconds to sleep thread.
216   */
217  private static void goToSleep(int timeInMilliseconds) {
218    // Sleep the current thread for amount passed so user can read
219    try {
220      Thread.sleep(timeInMilliseconds);
221    } catch (InterruptedException e) {
222      System.out.println("---- CANNOT SLEEP THREAD -----");
223    }
224  }
225
226  /**
227   * This method shows the usage of Inheritance and Polymorphism.
228   */
```

```java
@SuppressWarnings("resource") //for scanner not closing see below for why
private static void goToTheStore() {

  // Inheritance is extending a class from a base class in order to "inherit"
  // Properties from the parent class.
  // Polymorphism is using an object to take on multiple objects. In this
  // case,
  // I'm using Clothes as a parent object and Pants/Shirt/Shoes as child
  // objects that run different variations
  // of the base class's methods. I'm also changing the base properties
  // through each child's constructor.

  Scanner scan = new Scanner(System.in);
  List<Clothes> myClothes = new ArrayList<Clothes>();

  boolean makingBigDecisions = true;
  do {
    System.out.println(
        "So.. I guess you're going to the store now. What will you wear?");
    System.out.println("Pants: 1");
    System.out.println("Shirt: 2");
    System.out.println("Shoes: 3");
    System.out.println("Leave for Store: 0");
    int userInput = 0;
    try {
      scan = new Scanner(System.in);
      userInput = scan.nextInt();
      scan.nextLine();
      // In case they didn't pick a correct choice
      if (userInput < 0 || userInput > 3) {
        // scan.close(); -> if I close the scanner - it will cause an infinite
        // loop
        throw new InputMismatchException();
      }
    } catch (InputMismatchException e) {
      System.out.println("That's not one of the choices! Try again.");
      continue;
    } catch (Exception e) {
      System.out.println("Catch-All Error. You messed something up!");
      System.out.println(e);
      continue;
    }
    switch (userInput) {
      case 0:
        makingBigDecisions = false;
        break;
      case 1:
        myClothes.add(new Pants());
        break;
      case 2:
        myClothes.add(new Shirt());
        break;
      case 3:
        myClothes.add(new Shoes());
        break;
      default:
        break;
```

```
286       }
287     } while (makingBigDecisions);
288
289     // At Store
290     System.out.println("\n");
291     scan.close();
292
293     goToStore(myClothes);
294
295     goToSleep(2000);
296   }
297
298   /**
299    * Method used from the main "goToTheStore()" Method Finds which clothes the
300    * user obtained.
301    *
302    * @param clothes
303    *          the list of Clothes from the User.
304    */
305   private static void goToStore(List<Clothes> clothes) {
306     boolean shoes = false;
307     boolean shirt = false;
308     boolean pants = false;
309
310     for (Clothes c : clothes) {
311       if (c.isBodyCovered()) {
312         shirt = true;
313       } else if (c.areLegsCovered()) {
314         pants = true;
315       } else if (c.areFeetCovered()) {
316         shoes = true;
317       }
318     }
319
320     if (shoes && shirt && pants) {
321       System.out.println("Good job! You can dress yourself!");
322     } else {
323       System.out.println("You fail at life.");
324     }
325   }
326
327   /**
328    * Random String Stuff.
329    */
330   private static void stringProcessing() {
331     System.out.println("Random String Stuff...\n");
332     String s1 = "Mike";
333     String s2 = "Ike";
334     String s3 = String.join("-n-", s1, s2);
335     System.out.println("\t" + s3 + "\n");
336
337     System.out.println("What is the meaning of life?");
338     String s = "12425";
339     try {
340       System.out.println(Integer.parseInt(s.substring(2, s.length() - 1)));
341     } catch (StringIndexOutOfBoundsException e) {
342       System.out.println("Out of Bounds Exception");
```

```
343     } catch (Exception ex) {
344       System.out.println("Bad String");
345       System.out.println(ex);
346     }
347
348     System.out.println();
349
350     String s4 = "Application Complete";
351     for (int i = 0; i < s4.length(); i++) {
352       if (s4.charAt(i) == ' ') {
353         System.out.println();
354         continue;
355       } else {
356         System.out.print(s4.charAt(i));
357       }
358     }
359   }
360 }
361
```

```java

/**
 * @author Warren Smith.
 *
 */
public abstract class Clothes {

  protected boolean bodyCovered = false;
  protected boolean legsCovered = false;
  protected boolean feetCovered = false;

  /**
   * Constructor.
   */
  public Clothes() {
    bodyCovered = false;
    legsCovered = false;
    feetCovered = false;
  }

  /**
   * @return Whether body is covered.
   */
  public boolean isBodyCovered() {
    return bodyCovered;
  }

  /**
   * @return Whether legs are covered.
   */
  public boolean areLegsCovered() {
    return legsCovered;
  }

  /**
   * @return Whether feet are covered.
   */
  public boolean areFeetCovered() {
    return feetCovered;
  }
}

```

```java
/**
 * @author Warren Smith.
 *
 */
public class Shirt extends Clothes {

  /**
   * Constructor.
   */
  public Shirt() {
    this.bodyCovered = true;
  }
}

```

```java
/**
 * @author Warren Smith.
 *
 */
public class Pants extends Clothes {

  /**
   * Constructor.
   */
  public Pants() {
    this.legsCovered = true;
  }
}

```

```java
/**
 * @author Warren Smith.
 *
 */
public class Shoes extends Clothes {

  /**
   * Constructor.
   */
  public Shoes() {
    this.feetCovered = true;
  }
}

```

```java

/**
 * @author Warren Smith This Class is used to retain coordinates for 2D Array.
 */
public class Vector2 {
  private int valueX;
  private int valueY;

  /**
   * Constructor.
   * @param x The X coordinate.
   * @param y The Y coordinate.
   */
  public Vector2(int x, int y) {
    valueX = x;
    valueY = y;
  }

  /**
   * Sets the X and Y values to zero.
   */
  public void zero() {
    valueX = valueY = 0;
  }

  /**
   * Does a print line of both the X and Y values.
   */
  public void print() {
    System.out.println("( " + valueX + ", " + valueY + " )");
  }
}

```