Exercise 5.2

The documentation contains the following sections:

- Overall description of the class: its purpose and how to use it
- A brief summary of the fields
- A brief summary of the constructors
- A brief summary of the methods
- A detailed description of the fields
- A detailed description of the constructors
- A detailed description of the methods

Exercise 5.3

Both check whether the string contains their first argument. The single-argument version of the method starts the match at index 0 within the string. The two-argument version starts the match at the position of the second argument. Both return true if there is a match and false otherwise.

Exercise 5.4

Yes:

```
public boolean endsWith(String suffix)
```

Exercise 5.5

Yes: `public int length()`

Exercise 5.7

Signature: `public String trim()`
Example: `String trimmedText = text.trim();`

Exercise 5.8

add the line:

```
input = input.trim();
```

to the start() method of SupportSystem

Exercise 5.9

add the line:

```
input = input.toLowerCase();
```

to the start() method of SupportSystem

Exercise 5.10

boolean

Exercise 5.11

```
if(input.equals("bye")) {
    finished = true;
}
```

Exercise 5.12

Package: java.util
It generates random numbers

An instance is created by using one of the two constructors:

```
Random random = new Random();
Random random = new Random(seed);
```

To generate a random integer:

```
random.nextInt();
```

Exercise 5.13

```
Random random = new Random();
random.nextInt();
```

Exercise 5.14

```
import java.util.Random;


public class RandomTester
{
    private Random random;  // the random number generator

    public RandomTester()
    {
        random = new Random();
    }

    public void printOneRandom()
    {
        System.out.println(random.nextInt());
    }

    public void printMultiRandom(int howMany)
    {
        for(int i = 0; i < howMany; i++) {
            printOneRandom();
        }
    }
}
```

Exercise 5.15

```
0,1,2,...,99
```

Exercise 5.16

```
public int throwDice()
{
    return random.nextInt(6) + 1;
}
```

Exercise 5.17

```
public String getResponse()
{
    int answer = random.nextInt(3);
    if(answer == 0) {
        return "yes";
    }
    else if(answer == 1) {
        return "no";
    }
    else {
        return "maybe";
    }

}
```

A more sophisticated alternative would be:

```
public String getResponse()
{
    String[] responses = {
        "yes",
        "no",
        "maybe",
    };

    return responses[random.nextint(responses.length)];
}
```

Exercise 5.18

```
private ArrayList<String> responses;

public RandomTester()   // constructor
{
    responses = new ArrayList<String>();
    responses.add("yes");
    responses.add("don't know");
    ...
}

public String getResponse()
{
    return responses.get(random.nextInt(responses.size()));
}
```

Exercise 5.19

```
    public int getOneRandom(int max)
    {
        return random.nextInt(max) + 1;
    }
```

Exercise 5.20

```
    public int getOneRandom(int min, int max)
    {
        return random.nextInt(max - min + 1) + min;
    }

    public int getOneRandom(int max)
    {
        return getOneRandom(1, max);
    }
```

Exercise 5.22

When you add more responses these also get shown randomly together with the existing ones.
This is because the length of the list with responses is used when generating the random number.

Exercise 5.24

Methods in HashMap that depend on the type parameters:

- Set<Map.Entry<K,V>> entrySet()
- V get(Object key)
- Set<K> keySet()
- V put(K key, V value)
- void putAll(Map<? extends K,? extends V> m)
- V remove(Object key)
- Collection<V> values()

Yes, the same type could be used for both parameters.

Exercise 5.25

Call its size() method.

Exercise 5.26

```
public class MapTester
{
    private HashMap<String, String> phoneBook = new HashMap<String,
String> ();

    public MapTester()
    {
        phoneBook.put("Homer Jay Simpson", "(531) 9392 4587");
        phoneBook.put("Charles Montgomery Burns", "(531) 5432 1945");
        phoneBook.put("Apu Nahasapeemapetilon", "(531) 4234 4418");
    }

    public void enterNumber(String name, String number)
    {
        phoneBook.put(name, number);
    }

    public String lookupNumber(String name)
    {
        return phoneBook.get(name);
    }
}
```

Exercise 5.27

It overwrites the previous value associated with the key.

Exercise 5.28

Both values stay in the map. HashMaps only uses the key to distinguish entries - not the values.

Exercise 5.29

phoneBook.containsKey("Homer Jay Simpson");

Exercise 5.30

It returns null.

Exercise 5.31 (duplicates 5.25)

```
phoneBook.size()
```

Exercise 5.32

Call the `keySet` method to return the `Set` of keys, and then iterate over the set.

```
for(String name : phoneBook.keySet()) {
    System.out.println(name);
}
```

Exercise 5.34

Similarities between HashSet and ArrayList
Both contains a collection of objects
It is possible to add objects (with the add method)
It is possible to remove objects (with the remove method)
Both have a size()
Both have provide an iterator() method to go through all the elements

Differences:
In a HashSet each object can only appear once in the set (because it is a Set). In a ArrayList an Object can appear multiple times.
An ArrayList is ordered while a HashSet is not ordered.

Exercise 5.35

You can use regular expression to define how a string should be split.

Some documentation on regular expressions in Java under the Pattern class in the java.util.regex package.

Exercise 5.36

```
String[] words = s.split("[ \t]");

String[] words = s.split(":");
```

Exercise 5.37

Using HashSet guaranties that there are no duplicate elements, but it does not keep the order of the words.

Exercise 5.38

It creates empty strings - which was probably not the intention. to fix it we could do this:

```
String[] words = s.split("[ \t]+");
```

Exercise 5.39

```
import java.util.Arrays;

The Arrays class defines various sort methods:

    public void sortValues(int[] values)
    {
        Arrays.sort(values);
        for(int value : values) {
            System.out.print(value + " ");
        }
        System.out.println();
    }
```

Exercise 5.40

See the modified start() method of SupportSystem in: Code 5.6
and the modified generateResponses() in Responder in: Code 5.7

Exercise 5.43

The modifications needed to the Responder class in the project tech-support-complete:

Add a new field:

```
    private HashMap<String, String> synonymMap;
```

Initialize it in the constructor:

```
    synonymMap = new HashMap<String, String>();
```

Add this to the generateResponse method right after the if-block:

```
    else {
        //check if it is a synonym
        String synonym = synonymMap.get(word);
        if(synonym != null) {
            return responseMap.get(synonym);
        }
    }
```

To create an example replace the `responseMap.put("crashes","Well....);` with: `synonymMap.put("crashes", "crash");`

Exercise 5.47

Keyword examples:

@author
@version
@param
@return

These keywords get special attention so they stand out in the documentation.

Exercise 5.53

- Color.BLUE is used to set the pen color in drawSquare()
- Color.RED is used in drawWheel()

Exercise 5.54

Other colors available in the Color class are BLACK, GREEN, MAGENTA, ORANGE, etc. These are available via both upper-case and lower-case names.

Exercise 5.56

Use the clear() method to clear the whole canvas.

Exercise 5.57

```
    /**
     * Draw a triangle on the screen.
     */
    public void drawTriangle()
    {
        Pen pen = new Pen(320, 260, myCanvas);
        pen.setColor(Color.GREEN);
        triangle(pen);
    }

    /**
     * Draw a triangle in the pen's color at the pen's location.
     */
    private void triangle(Pen pen)
    {
        for(int i = 0; i < 3; i++) {
            pen.move(100);
            pen.turn(120);
        }
    }
```

Exercise 5.58

```
    /**
     * Draw a pentagon on the screen.
     */
    public void drawPentagon()
    {
        Pen pen = new Pen(320, 260, myCanvas);
        pen.setColor(Color.MAGENTA);
        pentagon(pen);
    }

    /**
     * Draw a pentagon in the pen's color at the pen's location.
     */
    private void pentagon(Pen pen)
    {
        for(int i = 0; i < 5; i++) {
            pen.move(100);
            pen.turn(360 / 5);
        }
    }
```

Exercise 5.59

```
    /**
     * Draw a polygon with the given number of sides.
     * @param n The number of sides.
     */
    public void drawPolygon(int n)
    {
        Pen pen = new Pen(320, 260, myCanvas);
        pen.setColor(Color.MAGENTA);
        polygon(pen, n);
    }

    /**
     * Draw a polygon with the given number of side
     * in the pen's color at the pen's location.
     * @param sides The number of sides.
    */
    private void polygon(Pen pen, int sides)
    {
        for(int i = 0; i < sides; i++) {
            pen.move(100);
            pen.turn(360 / sides);
        }
    }
```

Exercise 5.60

See **05-60-spiral**:

```
    /**
     * Draw a spiral in the pen's color at the pen's location.
     */
    private void spiral(Pen pen)
    {
        // The number of arms.
        int arms = 63;
        // The current length of the arm being drawn.
        int armLength = 3;
        // How much longer to make each arm.
        int armIncrement = 2;

        // Start in the middle.
        pen.penUp();
        Dimension size = myCanvas.getSize();
        pen.moveTo(size.width / 2, size.height / 2);
        // Face downwards.
        pen.turnTo(90);
        pen.penDown();

        // Draw arms of increasing length.
        for(int arm = 0; arm < arms; arm++) {
            pen.move(armLength);
            pen.turn(90);
            armLength += armIncrement;
        }
    }
```

Exercise 5.62

```
    public void bounce(int numberOfBalls)
    {
        int ground = 400;   // position of the ground line
        myCanvas.setVisible(true);
        // draw the ground
        myCanvas.drawLine(50, ground, 550, ground);
        // create and show the balls
        HashSet<BouncingBall> balls = new HashSet<BouncingBall>();
        for(int i=0; i < numberOfBalls; i++) {
            BouncingBall ball = new BouncingBall(50+32*i, 50, 16,
Color.blue, ground, myCanvas);
            balls.add(ball);
            ball.draw();
        }
        // make them bounce
        boolean finished =  false;
        while(!finished) {
            myCanvas.wait(50);           // small delay
            for(BouncingBall ball : balls) {
                ball.move();
                // stop once ball has travelled a certain distance on
x axis
                if(ball.getXPosition() >= 550 + 32*numberOfBalls) {
                    finished = true;
                }
            }
        }
        for(BouncingBall ball : balls) {
            ball.erase();
        }
    }
```

Exercise 5.63

HashSet is most suitable, because it guaranties that we only have one of each ball in the collection. The HashMap could be used for this as well, but we do not need a map, so it would be a bad choice.

Exercise 5.64

```
        // create and show the balls
        Random random = new Random();
        HashSet balls = new HashSet<BouncingBall>();
        for(int i=0; i < numberOfBalls; i++) {
            Dimension size = myCanvas.getSize();
            int x = random.nextInt((int) size.getWidth());
            int y = random.nextInt((int) size.getHeight());
            BouncingBall ball = new BouncingBall(x, y, 16,
Color.blue, ground, myCanvas);
            balls.add(ball);
            ball.draw();
        }
```

Exercise 5.65+5.66

Download: 05-65-balls-inabox.zip

Exercise 5.68

```
public static final double TOLERANCE = 0.001;

private static final int PASS_MARK = 40;

public static final char HELP_COMMAND = 'h';
```

Exercise 5.69

Five constants are defined. They are used to index the positions of data in an array. It is a good use of constants.

Exercise 5.70

You would only have to modify the values of the constants which are all defined in one place. If these numbers had not been placed in constant fields, but instead used directly, it would have required changes to several different parts of the code.

Exercise 5.71

```
public class NameGenerator
{
    public String generatorStarWarsName(String firstName, String 
lastName,
                                        String mothersMaidenName, 
String cityOfBirth)
    {
        String swFirstName = lastName.substring(0,3) + 
firstName.substring(0,2);
        String swLastName = mothersMaidenName.substring(0,2) + 
cityOfBirth.substring(0,3);
        return swFirstName + " " + swLastName;
    }
}
```

Exercise 5.72

Strings are immutable and therefore can not be changed. The method that is called does not change the instance 's' but returns a new object with the string in upper case. The correct way to do it is:

```
String upperCaseS = s.toUpperCase();
System.out.println(upperCaseS);
```

Exercise 5.73

The variable a and b contains values. When these values are passed as arguments to the method, *the values* get copied into the variables i1 and i2. So we now have two copies of the values in a and b. In the method, we then swap the values stored in the variables i1 and i2. This has no effect outside the method as i1 and i2 are local variables in the method. After calling the method the variables a and b will still contain the same values as before the method call.