Chapter 6

6.1: opened and used some of the project

6.2: The String class represents all character strings. Classes in the java library have lists of constructors and what they can do as well as methods and summaries of what they do and some examples of using the String class

6.3: The startsWith method tests to see of the start of a string or specified index value uses a specific prefix. The differences are each startsWith method has different parameters which determine where and how is looks for a specific prefix. One of the methods has an int parameter to specify index value and the other doesn't.

6.4: endsWith is a method that looks for a String suffix and tests id a string ends with a specific suffix.

6.5: the length method returns the length of this string

6.6: Looking through the java api and the available methods it is pretty easy to find and learn how to use a given method

6.7: public String trim()

You could have a string with empty spaces that you want gone

String text = “ Blake Soderlund “

System.out.println(text.trim());

Would return “Blake Soderlund” without the spaces on the end

6.8/6.9/6.11:

```
public void start()
{
    boolean finished = false;

    printWelcome();

    while(!finished) {
        String input = reader.getInput().trim().toLowerCase();

        if(input.equals("bye")) {
            finished = true;
        }
        else {
            String response = responder.generateResponse();
            System.out.println(response);
        }
    }

    printGoodbye();
}
```

6.10: The return type is Boolean so either true or false

6.12: the Random class is in the java.util package. The class is used to create a stream of pseudorandom numbers. You can create a new random number generator without a parameter or crease one using a single long seed Random(long seed).

6.13:
```
public static int randomInt(int min, int max)
{
   Random rnd = new Random();
   int num = rnd.nextInt((max - min)+ 1)+ min;
   return num;
}
```

6.14:
```
import java.util.Random;
public class RandomTester
{
   private Random rnd;
   /**
    * Constructor for objects of class RandomTester
    */
   public RandomTester()
   {
      rnd = new Random();
   }

   public void printOneRandom(int num)//parameter for range of numbers
   {
      // put your code here
      int index = rnd.nextInt(num);
      System.out.println(index);
   }

   public void printMultiRandom(int howMany)
   {
        while(0 < howMany)
      {
        int index = rnd.nextInt(100);//set the possible random numbers to 0-99
        System.out.println(index);
        howMany = howMany - 1;
      }
   }
}
```

6.15:
```
public static void printOneRandom()
{
   Random Rnd = new Random();
   int index = Rnd.nextInt(100); //range of 0 - 99
   System.out.println(index);
}
```
Prints a random number from 0 - 99

6.16:
```
public static void throwDie()
{
   int rndDie = 6;
   Random RND = new Random();
   while(0 < rndDie)
   {
      int die = RND.nextInt(6) + 1;
      System.out.println(die);
      rndDie--;
   }
}
```
Displays 6 random numbers between 1 and 6 (inclusive)

6.17:
```
public static void getResponse()
{
   Random response = new Random();
   int x = response.nextInt(3);
   if(x == 0){
      System.out.println("yes");
   }
   if(x == 1){
      System.out.println("no");
   }
   if(x == 2){
      System.out.println("maybe");
   }
}
```

6.18:
```
import java.util.Random;
import java.util.ArrayList;

//Start off by importing what i need
```

```
public class RandomTester
{
   private Random rnd;
   private ArrayList<String> responses;
   //declare fields

   public RandomTester()
   {
       rnd = new Random();
       responses = new ArrayList<>();
       fillResponses();
    }
    //call a method to fill the array list with some default responses

   //Creates a random number from the amount of responses in the arraylist
    public String getResponse()
    {
       int index = rnd.nextInt(responses.size());
       return responses.get(index);
    }
   //fills arraylist when the constructor method is called
    private void fillResponses()
    {
       responses.add("Yes");
       responses.add("No");
       responses.add("Maybe");
    }
}
```

6.19:
```
public int returnRandom(int max)
{
   int rando = rnd.nextInt(max) + 1;
   return rando;
}
```

6.20:
```
public int getRandom(int min, int max)
{
   return rnd.nextInt(max - min + 1) + min;
}
```

6.21: secure random creates a string of random numbers which is good for security because these strings are unlikely to be created manually if they are long enough and random
6.23: Adding in more response adds them to the current list because the method uses the lenth of the list of responses to get the random number.

6.24: A hash map is a hash table based on javas map interface which is a collection of key-value pairs. So its a collection that maps keys to values

6.25: Set<Map.Entry<K,V>> entrySet()

V get(Object key)

V put (K key, Value)

V remove(object key)

Collection<V> values ()

Yes ,the same type could be used for both parameters

6.26: by calling its size() method

6.27:

```
import java.util.HashMap;
public class MapTester
{
   private HashMap<String, String> contacts = new HashMap<> ();
   public MapTester()
   {
      contacts.put("Charles Nguyen", "(531) 9392 4587");
      contacts.put("Lisa Jones", "(402) 4536 4674");
      contacts.put("William H. Smith", "(998) 5488 0123");
   }
   public void enterNumber(String name, String number)
   {
     contacts.put(name, number);
   }
   public String lookupNumber(String name)
   {
      return contacts.get(name);
   }
}
```

6.28: it replaces the previous value tied to the key

6.29: Both values can be in the map because they keys are what determine entries

6.30: contacts.containsKey(“Lisa Jones”);

6.31: will return null

6.32: Use the keySet method to iterate over the set of keys

```
for(String name : contacts.keySet() )
{
system.out.println(name);
}
```

6.33:

6.34: Both have a size() method, provide an iterator method, allow the addition or removal of objects, store any number of objects

A HashSet isnt ordered
A HashSet can only have an object appear once

6.35: By using an expression \
6.36: String[] wordArray = input.line("\\s+");
String[] wordArray = input.line(":");
6.37: HashSet can not store duplicate objects but isnt ordered
6.38: It could create empty strings so the solution is
String[] wordArray = input.line("\\s+");
Which removes all whitespace
6.39:
6.40: The Array class as all sorts of methods to manipulate arrays such as toString(), sort(), fill(), hashCode, equals(), binarySearch(), asList(), copyOf() and some more variants on those
6.41:
6.42 6.43 6.44: Code implemented

6.45:
We would need to make a new field and have the constructor initialize it

private HashMap<String, String> synonyms:

synonyms = new HashMap<>();

We would add synonyms to the map

synonyms.put("crash","crashes","crashed");
synonyms.put("buggy","bugs","bugged");

Then check for them in the generateResponse method

6.46:
6.47:

6.48: putIfAbsent checks of the specified key is associated with the given value or null. Then it associates it with the given and returns null or the current value
6.49:
6.50:
6.51:

6.52:
6.53:
6.54:

6.55: Draw demo has 4 methods the first two draw specified shapes at specific places on the canvas. The Square method uses the Pen class to draw a line with predetermined length and turn 90 degrees and repeat this 4 times to draw a square. The wheel is similar but a more complex shape. The colorScribble draws three lines each a different color and at random length and directions for a scribble look. The clear erases all marks on the canvas
6.56 6.57:pen and canvas classes investigated

6.58: drawSquare() uses Color.BLUE to specify color
drawWheel() uses Color.RED to specify color
6.59: other colors in the Color class black,blue, cyan, orange, pink and more with uppercase variants
6.60: red circle drawn
6.61: the clear() method will clear the canvas
6.62:

```
public void drawTriangle()
{
    Pen pen = new Pen(150, 200, myCanvas);
    pen.setColor(Color.GREEN);

    for (int i = 0; i<3; i++)
        {
            pen.move(75);
            pen.turn(120);
        }
}
```

6.63:

```
public void drawPentagon()
{
    Pen pen = new Pen(250, 200, myCanvas);
    pen.setColor(Color.black);
    for (int i=0; i<5; i++) {
        pen.move(100);
        pen.turn(72);
    }
}
```

6.64:
```
public void drawPolygon(int n)
{
    Pen pen = new Pen(150, 100, myCanvas);
    pen.setColor(Color.GREEN);
    for (int i=0; i<n; i++) {
        pen.move(30);
        pen.turn(360/n);
    }
}
```

6.65:
```
public void spiral()
{
    Pen pen = new Pen(250, 200, myCanvas);
    pen.setColor(Color.black);
    //set the spiral dimensions
    int segments = 50;
    int segmentLength = 2;
    int segmentIncrement = 1;
    for(int segment = 0; segment < segments;)
    {
        pen.move(segmentLength);
        segmentLength += segmentIncrement;
        pen.turn(60);
    }
}
```

6.66: 19 methods are shown

6.67:
```
public void noPenPic()
{
    myCanvas = new Canvas("Drawing Demo", 500, 400);
    //makes a triangle
    myCanvas.drawLine(100, 150, 200, 250);
    myCanvas.drawLine(200, 250, 250, 200);
    myCanvas.drawLine(250, 200, 100, 150);
}
```

```
6.68:       public void bounce(int numberOfBalls)
            {
               int ground = 400;   // position of the ground line

               myCanvas.setVisible(true);

               // draw the ground
               myCanvas.setForegroundColor(Color.BLACK);
               myCanvas.drawLine(50, ground, 550, ground);

               // create and show the ball, if the new balls dont have different starting positions
            they cant be seen
               HashSet<BouncingBall> balls = new HashSet<>();
               for(int i=0; i<numberOfBalls; i++)
               {
                  BouncingBall ball = new BouncingBall(50+10*i, 50+8*i, 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();
                     if(ball.getXPosition() >= 550)
                     {
                        finished = true;
                     }
                  }
               }
               for(BouncingBall ball : balls)
                  {
                     ball.erase();
                  }
            }
         }
```

6.69: HashMap or HashSet make more sense than an arraylist because there can only be one of each ball in either collection. Between HashMap and HashSet the Hashset makes more sense because we don't need a map

6.70:
```
HashSet<BouncingBall> balls = new HashSet<>();
Random random = new Random();
for(int i=0; i<numberOfBalls; i++)
{
   BouncingBall ball = new BouncingBall(random.nextInt(500), random.nextInt(25),
16, Color.BLUE, ground, myCanvas);
   balls.add(ball);
   ball.draw();
}
```

Imported the random class and set the x and y values to be within the range that was needed

6.71:

6.72:

6.73:

6.74:

```
public static final double tolerance = 0.001;
private static final int passMark = 40;
public static final char helpCommand = ‘h’;
```

6.75:java.lang.math defines both pi and e

6.76: If its in 10 different places than associating it with a variable name means that if it needs to be changed than you will only need to change it once and it will insure all instances will be the same. With a name its importance will be easier to determine by others reading the code.

6.77: static int max(int a, int b);

6.78: methods in the math class are static because they are mathematical operations and don't change based on an object's current state. You could use these methods as instance methods if you create an instance of the math class

6.79:
```
public static long countToOneThousand()
{
   long x = 0;
   long start = System.currentTimeMillis();
   for(long i=1; i<=10000; i++){
      x++;
      System.out.println(x);
   }
   long end = System.currentTimeMillis();
   return end - start;

}
}
```

6.80: a static method can be called from an instance method
An instance method can be called from a static method if there is a reference to an instance to call on like the exercise above.
A static method can be called from a static method.
6.81: Collection class read
6.82:
```
public static void main(String[] args)
{
   SupportSystem system = new SupportSystem();
   system.start();
}
```

6.83:
6.84: