More Sophisticated Behavior

• In this chapter there will be two projects
  – The first project is a TechSupport System, there are three versions of increasing complexity
  – The second project, “Balls”, returns to our earlier Canvas class and develops an animation
• The major new topics are
  – Random number generation
  – HashMap and HashSet, two new collections
  – StringTokenizer for analyzing input
  – Information hiding and coupling
  – Using JavaDoc to document your program
  – Class Implementation vs. Interface
Conversations with a Human

• The Turing Test, proposed in 1953, said that a computer would be intelligent if a human communicating with another “entity” the human could not distinguish if the entity was a computer or a human

• Joseph Weizenbaum developed a program in the 1960s called “Eliza” that appeared to carry on an intelligent conversation with a human

• Our first project, a technical support system, lets a human describe a computer problem to this software system and attempts to carry on a dialog that tries to determine the problem; this program is simplistic but shows some interesting techniques such as spotting keywords and responding accordingly
/**
 * This class implements a technical support system. It is the top
 * level class in this project. The support system communicates via
 * text input/output in the text terminal.
 */

public class SupportSystem
{
    private InputReader reader;
    private Responder responder;

    /**
     * Creates a technical support system.
     */
    public SupportSystem()
    {
        reader = new InputReader();
        responder = new Responder();
    }

    // methods go here
}
/**
 * Start the technical support system. This will print a welcome
 * message and enter into a dialog with the user, until the user
 * ends the dialog.
 */

public void start()
{
    boolean finished = false;

    printWelcome();

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

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

    printGoodbye();
}
A Typical Loop Structure

- Sometimes in the middle of the loop you may need to exit the loop under some circumstances but under other circumstances you want to continue

```java
boolean finished = false;
while(!finished)
    << do something here >>
    if(<<exit condition>> {  
        finished = true;
    } else {
        << do something more >>
    }
} // end while
```

This situation often occurs when there is interaction with the user during the initial phase of the loop and the user does something that indicates the loop should exit.
Greetings and Salutations

/**
 * Print a welcome message to the screen.
 */
private void printWelcome()
{
    System.out.println("Welcome to the DodgySoft Technical Support System.");
    System.out.println();
    System.out.println("Please tell us about your problem.");
    System.out.println("We will assist you with any problem you might have.");
    System.out.println("Please type 'bye' to exit our system.");
}

/**
 * Print a good-bye message to the screen.
 */
private void printGoodbye()
{
    System.out.println("Nice talking to you. Bye...");
}
import java.io.BufferedReader;
import java.io.InputStreamReader;
/**
 * Class InputReader reads typed text input from the standard text terminal.
 * The text typed by a user is returned as a string.
 */
public class InputReader
{
    private BufferedReader reader;
    /**
     * Create a new InputReader that reads text from the text terminal.
     */
    public InputReader()
    {
        reader = new BufferedReader(new InputStreamReader(System.in));
    }
    /**
     * Read a line of text from standard input and return it
     * @return  A single String of what the user typed
     */
    public String getInput()
    {
        System.out.print(">"); // print prompt
        String inputLine = readInputLine();
        return inputLine;
    }
}
/**
 * Read one line of input and return it as a String.
 * @return A Strings representing the input, or an empty String if an error occurs.
 */
private String readInputLine()
{
    String line = "";

    try {
        line = reader.readLine();
    }
    catch(java.io.IOException exc) {
        System.out.println ("There was an error during reading: " + exc.getMessage());
    }
    return line;
}
Buffered Reader

- We discuss data streams for input and output
- We have already seen standard output in printing to a text terminal using `System.out.println`
- The standard input stream, `System.in`, is the keyboard; this is accessed by creating an input stream reader
  
  ```java
  new InputStreamReader(System.in)
  ```

- This needs to be buffered by another reader as shown by the statement with two constructors
  
  ```java
  reader = new BufferedReader(
      new InputStreamReader(System.in))
  ```

- The `readLine` method fetches input a line at a time
Handling Exceptions

• Some statements in Java, such as input/output statements, require enclosure in an exception handler in case an exception occurs.

• The general form is:
  
  ```java
  try {
    <<statements that could cause an exception>>
  } catch (Exception e) {
    <<handle the exception here>>
  }
  ```

• The exception handler can be empty, but the try … catch statement is required.
/**
 * The responder class represents a response generator object. It is
 * used to generate an automatic response.
 *
 */

public class Responder {

    /**
     * Construct a Responder - nothing to do
     */
    public Responder() {
    }

    /**
     * Generate a response.
     * @return A string that should be displayed as the response
     */
    public String generateResponse() {
        return "That sounds interesting. Tell me more...";
    }
}

Our First Improvement

• In our first version, the response was always the same, hardly a sign of intelligence

• In our second version, we store several possible responses in an ArrayList and randomly select one of these responses

• This improves the conversation but does not directly respond to the user’s input concerning a technical problem

• This improvement introduces a new Java tool, a random number generator
import java.util.ArrayList;
import java.util.Random;

/**
 * This is the second version of this class. This time, we generate
 * some random behavior by randomly selecting a phrase from a predefined
 * list of responses.
 */
public class Responder
{
    private Random randomGenerator;
    private ArrayList responses;
    /**
     * Construct a Responder
     */
    public Responder()
    {
        randomGenerator = new Random();
        responses = new ArrayList();
        fillResponses();
    }

    // methods go here
}
A More Sophisticated Responder - 2

/**
 * Generate a response.
 *
 * @return A string that should be displayed as the response
 * *
 */
public String generateResponse()
{
    // Pick a random number for the index in the default response
    // list. The number will be between 0 (inclusive) and the size
    // of the list (exclusive).
    int index = randomGenerator.nextInt(responses.size());
    return (String) responses.get(index);
}
/**
 * Build up a list of default responses from which we can pick one
 * if we don't know what else to say.
 */
private void fillResponses()
{
    responses.add("That sounds odd. Could you describe that problem in more detail?");
    responses.add("No other customer has ever complained about this before. \n" +
                   "What is your system configuration?");
    responses.add("That sounds interesting. Tell me more...");
    responses.add("I need a bit more information on that.");
    responses.add("Have you checked that you do not have a dll conflict?");
    responses.add("That is explained in the manual. Have you read the manual?");
    responses.add("Your description is a bit wishy-washy. Have you got an expert\n" +
                   "there with you who could describe this more precisely?");
    responses.add("That's not a bug, it's a feature!");
    responses.add("Could you elaborate on that?");
}
Random Number Generation

• You must import the Random library:
  ```java
  import java.util.Random;
  ```

• Then you need to declare an object of type Random:
  ```java
  private Random randomGenerator;
  ```

• There are a number of methods you can call on a Random object; consult the online documentation to view these methods

• The method we need is to generate an integer between 0 and up to, but not including, an ending value, as seen here
  ```java
  int index = randomGenerator.nextInt(responses.size());
  ```
The Complete Tech Support System

• We need to analyze the user input in order to respond directly to that input
  – The input String needs to be “tokenized” into words to later see in the responder if a word is recognized
  – In Java, a StringTokenizer splits a string into words
  – These words are stored in a HashSet, another collection in Java

• Changes to the responder system
  – A set of keywords have been store in a HashMap along with the appropriate response to each keyword
  – If the keyword is found in the HashSet, the corresponding response is generated; otherwise a random response is generated
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.StringTokenizer;
import java.util.Set;
import java.util.HashSet;

/**
 * Class InputReader reads typed text input from the standard text terminal.
 * The text typed by a user is then chopped into words, and a set of words
 * is provided.
 */

public class InputReader
{
    private BufferedReader reader;

    /**
     * Create a new InputReader that reads text from the text terminal.
     */
    public InputReader()
    {
        reader = new BufferedReader(new InputStreamReader(System.in));
    }

    // methods go here
}
public HashSet getInput() {
    System.out.print("> ");               // print prompt
    String inputLine = readInputLine().trim().toLowerCase();
    StringTokenizer tokenizer = new StringTokenizer(inputLine);
    HashSet words = new HashSet();
    while (tokenizer.hasMoreTokens())
        words.add(tokenizer.nextToken()); // add each word to the set
    return words;
}

// private String readInputLine()
// this methods has not changed and is not repeated here
HashSet

• A mathematical set is an unordered collection of objects; a HashSet in Java is similar to this mathematical concept
• Our objects will be Strings that correspond to words in the sentence
• The method `add(<object>)` puts an object into a HashSet
• The get methods will be used by the responder to check if an item is in the HashSet or not
StringTokenizer

• The class StringTokenizer divides up a String value into separate values based on a set of separator characters

        StringTokenizer tokenizer = new StringTokenizer(inputLine);

• In a while loop you first check if there are more tokens

        tokenizer.hasMoreTokens()

• If there are more tokens, you fetch the next token

        tokenizer.nextToken()

• More documentation can be found online at www.java.sun.com
import java.util.HashMap;
import java.util.HashSet;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;

/**
 * The responder class represents a response generator object.
 * It is used to generate an automatic response, based on specified input.
 * Input is presented to the responder as a set of words, and based on those
 * words the responder will generate a String that represents the response.
 * 
 * Internally, the responder uses a HashMap to associate words with response
 * strings and a list of default responses. If any of the input words is found
 * in the HashMap, the corresponding response is returned. If none of the input
 * words is recognized, one of the default responses is randomly chosen.
 *
 */
public class Responder
{
    private HashMap responseMap;       // used to map key words to responses
    private ArrayList defaultResponses; // default responses if we don't recognise a word
    private Random randomGenerator;

    public Responder()
    {
        responseMap = new HashMap();
        defaultResponses = new ArrayList();
        fillResponseMap();
        fillDefaultResponses();
        randomGenerator = new Random();
    
}
/**
 * Generate a response from a given set of input words.
 *
 * @param words  A set of words entered by the user
 * @return       A string that should be displayed as the response
 */

public String generateResponse(HashSet words)
{
    Iterator it = words.iterator();
    while(it.hasNext()) {
        String word = (String) it.next();
        String response = (String) responseMap.get(word);
        if(response != null) {
            return response;
        }
    }
    // If we get here, none of the words from the input line was recognized.
    // In this case we pick one of our default responses (what we say when
    // we cannot think of anything else to say...)

    return pickDefaultResponse();
}
/**
 * Enter all the known keywords and their associated responses
 * into our response map.
 */
private void fillResponseMap()
{
    responseMap.put("crash",
        "Well, it never crashes on our system. It must have something\n        to do with your system. Tell me more about your configuration.");
    responseMap.put("crashes",
        "Well, it never crashes on our system. It must have something\n        to do with your system. Tell me more about your configuration.");
    responseMap.put("slow",
        "I think this has to do with your hardware. Upgrading your processor\n        should solve all performance problems. Have you got a problem with\n        our software?");
    responseMap.put("performance",
        "Performance was quite adequate in all our tests. Are you running\n        any other processes in the background?");
    responseMap.put("bug",
        "Well, you know, all software has some bugs. But our software engineers\n        are working very hard to fix them. Can you describe the problem a bit\n        further?");
    responseMap.put("buggy",
        "Well, you know, all software has some bugs. But our software engineers\n        are working very hard to fix them. Can you describe the problem a bit\n        further?");
responseMap.put("windows",
   "This is a known bug to do with the Windows operating system. Please\n   report it to Microsoft. There is nothing we can do about this.");
responseMap.put("macintosh",
   "This is a known bug to do with the Mac operating system. Please\n   report it to Apple. There is nothing we can do about this.");
responseMap.put("expensive",
   "The cost of our product is quite competitive. Have you looked around\n   and really compared our features?");
responseMap.put("installation",
   "The installation is really quite straight forward. We have tons of\n   wizards that do all the work for you. Have you read the installation\n   instructions?");
responseMap.put("memory",
   "If you read the system requirements carefully, you will see that the\n   specified memory requirements are 1.5 giga byte. You really should\n   upgrade your memory. Anything else you want to know?");
responseMap.put("linux",
   "We take Linux support very seriously. But there are some problems.\n   Most have to do with incompatible glibc versions. Can you be a bit\n   more precise?");
responseMap.put("bluej",
   "Ahhh, BlueJ, yes. We tried to buy out those guys long ago, but\n   they simply won't sell... Stubborn people they are. Nothing we can\n   do about it, I'm afraid.");
}
private void fillDefaultResponses()
{
    defaultResponses.add("That sounds odd. Could you describe that problem in more detail?\n");
    defaultResponses.add("No other customer has ever complained about this before. \n" + 
                        "What is your system configuration?\n");
    defaultResponses.add("That sounds interesting. Tell me more...\n");
    defaultResponses.add("I need a bit more information on that.\n");
    defaultResponses.add("Have you checked that you do not have a dll conflict?\n");
    defaultResponses.add("That is explained in the manual. Have you read the manual?\n");
    defaultResponses.add("Your description is a bit wishy-washy. Have you got an expert\n" + 
                        "there with you who could describe this more precisely?\n");
    defaultResponses.add("That's not a bug, it's a feature!\n");
    defaultResponses.add("Could you elaborate on that?\n");
}

private String pickDefaultResponse()
{
    // Pick a random number for the index in the default response list.
    // The number will be between 0 (inclusive) and the size of the list (exclusive).
    int index = randomGenerator.nextInt(defaultResponses.size());
    return (String) defaultResponses.get(index);
}
HashMap

• As a mathematical object, a map is a correspondence between a unique key and a corresponding value
• In our application we will map a single word, a String, into a response, also a String, that will be printed for the user
• The put method places the key and corresponding value in the HashMap; the get method retrieves a value based on its key
• If no item has the specified key value, null is returned
Lab 12

• Build your own conversation system
• First choose your domain area; select something you are interested in and have special expertise
• Select a set of keywords that might be in the user input that you can recognize and respond to
• For each of these words, compose an appropriate response; enter these into the HashMap in the order of importance
• Then design a set of default responses in case no keywords are recognized; try to coax the user into a response that may contain a keyword
JavaDoc Documentation

• Choose the names of all objects, fields, methods, and so forth to clearly reflect their meaning

• As a minimum you should document
  – The overall purpose of the class
  – Author and version information
  – A description of each constructor and method including
    • The method name and return type
    • The parameter names and types
    • A description of the purpose of each function, including each parameter and the return value
  – In describing methods, tell what they do so a user can use the method; do NOT describe how it works
Information Hiding

• In general, fields should be private
  – Use an accessor method if the value must be retrieved
  – Use a mutator if the value must be changed
  – Constant fields are an exception and should be public

• In general, methods are public
  – They are used to examine or change objects in a class
  – Methods that are tools for other methods in the same class should be private

• The general rule is only make public what must be public and keep everything else private
Class Implementation vs. Interface

• Classes typically include fields, constructors, methods including parameter list and method body, and appropriate documentation.
• Interfaces are similar, but the methods only have a parameter list and no implementation body
• This means that anyone who uses an Interface is obliged to complete the method implementations
• Interfaces abstract the essential information to use the class effectively
• An Interface may be an abstraction of the behavior of two different implementations, such as List being an Interface and ArrayList and LinkedList being two different implementations
The Balls Project

- The methods draw objects on a Canvas
- The method bounce() is an animation of two balls bouncing from left to right across the canvas; the gravity can be adjusted
- The method drawDemo() shows how to draw lines, text, and geometric objects
- This project uses
  - "final" to specify constant fields
  - "static" to specify methods or fields that can only have one copy
import java.awt.*;
import java.awt.geom.*;

/**
 * Class BallDemo - provides two short demonstrations showing how to use the
 * Canvas class.
 */

public class BallDemo
{
    private Canvas myCanvas;

    /**
     * Create a BallDemo object. Creates a fresh canvas and makes it visible.
     */
    public BallDemo()
    {
        myCanvas = new Canvas("Ball Demo", 600, 500);
        myCanvas.setVisible(true);
    }

    // methods go here
}
public void drawDemo()
{
    myCanvas.setFont(new Font("helvetica", Font.BOLD, 14));
    myCanvas.setForegroundColor(Color.red);
    myCanvas.drawString("We can draw text, ...", 20, 30);
    myCanvas.wait(1000);
    myCanvas.setForegroundColor(Color.black);
    myCanvas.drawString("...draw lines...", 60, 60);
    myCanvas.wait(500);
    myCanvas.setForegroundColor(Color.gray);
    myCanvas.drawLine(200, 20, 300, 50);
    myCanvas.wait(500);
    myCanvas.setForegroundColor(Color.blue);
    myCanvas.drawLine(220, 100, 370, 40);
    myCanvas.wait(500);
    myCanvas.setForegroundColor(Color.green);
    myCanvas.drawLine(290, 10, 320, 120);
    myCanvas.wait(1000);
    myCanvas.setForegroundColor(Color.gray);
    myCanvas.drawString("...and shapes!", 110, 90);
    myCanvas.setForegroundColor(Color.red);
    // the shape to draw and move
    int xPos = 10;
    Rectangle rect = new Rectangle(xPos, 150, 30, 20);
    // move the rectangle across the screen
    for(int i = 0; i < 200; i++) {
        myCanvas.fill(rect);
        myCanvas.wait(10);
        myCanvas.erase(rect);
        xPos++;
        rect.setLocation(xPos, 150);
    }
    myCanvas.fill(rect);
}
public void bounce()
{
    int ground = 400;   // position of the ground line

    myCanvas.setVisible(true);

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

    // crate and show the balls
    BouncingBall ball = new BouncingBall(50,50,16, Color.blue, ground, myCanvas);
    ball.draw();
    BouncingBall ball2 = new BouncingBall(70,80,20, Color.red, ground, myCanvas);
    ball2.draw();

    // make them bounce
    boolean finished = false;
    while(!finished) {
        myCanvas.wait(50);           // small delay
        ball.move(); ball2.move();
        // stop once ball has travelled a certain distance on x axis
        if(ball.getXPosition() >= 550 && ball2.getXPosition() >= 550)
            finished = true;
    }
    ball.erase();
    ball2.erase();
}
import java.awt.*;
import java.awt.geom.*;

/**
 * Class BouncingBall - a graphical ball that observes the effect of gravity.
 * The ball has the ability to move. Details of movement are determined
 * by the ball itself. It will fall downwards, accelerating with time due
 * to the effect of gravity, and bounce upward again when hitting the ground.
 */

public class BouncingBall
{
    private static final int gravity = 3;  // effect of gravity

    private int ballDegradation = 2;
    private Ellipse2D.Double circle;
    private Color color;
    private int diameter;
    private int xPosition;
    private int yPosition;
    private final int groundPosition;      // y position of ground
    private Canvas canvas;
    private int ySpeed = 1;                // initial downward speed
/**
 * Constructor for objects of class BouncingBall
 *
 * @param xPos  the horizontal coordinate of the ball
 * @param yPos  the vertical coordinate of the ball
 * @param ballDiameter  the diameter (in pixels) of the ball
 * @param ballColor  the color of the ball
 * @param groundPos  the position of the ground (where the wall will bounce)
 * @param drawingCanvas  the canvas to draw this ball on
 */

public BouncingBall(int xPos, int yPos, int ballDiameter, Color ballColor,
                     int groundPos, Canvas drawingCanvas)
{
    xPosition = xPos;
    yPosition = yPos;
    color = ballColor;
    diameter = ballDiameter;
    groundPosition = groundPos;
    canvas = drawingCanvas;
}

/**
 * Draw this ball at its current position onto the canvas.
 **/

public void draw()
{
    canvas.setForegroundColor(color);
    canvas.fillCircle(xPosition, yPosition, diameter);
}
/**
 * Erase this ball at its current position.
 **/
public void erase()
{
    canvas.eraseCircle(xPosition, yPosition, diameter);
}

/**
 * Move this ball according to its position and speed and redraw.
 **/
public void move()
{
    // remove from canvas at the current position
    erase();
    // compute new position
    ySpeed += gravity;
    yPosition += ySpeed;
    xPosition +=2;

    // check if it has hit the ground
    if(yPosition >= (groundPosition - diameter) && ySpeed > 0) {
        position = (int)(groundPosition - diameter);
        ySpeed = -ySpeed + ballDegradation;
    }

    // draw again at new position
    draw();
}
/**
 * return the horizontal position of this ball
 */
public int getXPosition()
{
    return xPosition;
}

/**
 * return the vertical position of this ball
 */
public int getYPosition()
{
    return yPosition;
}
Lab 13

- This lab will let you build a project from scratch
- The goal is to build a system that encrypts a String and decrypts a String
- The encryption and decryption will be based on a letter by letter substitution
- Random numbers will be used to generate the substitutions for both encryption & decryption
- These substitutions will be stored in HashMaps
- This goes beyond simply modifying an existing project, but, don’t worry, you will be given plenty of guidance to complete the project