CSS 142

Lecture 5

Arkady Retik aretik@uw.edu

TODAY'S CONTENT

1. **Feedback on HW1 and FoR**
2. **Scanner: continue from L4b (IO, Print)**
3. **Flow of Control: intro**
   - **Booleans; Branching**
4. **HW2 is due on Thur | check the rubric | HW3 is next.**
5. **Activity in class (on I/O; Scanner)**

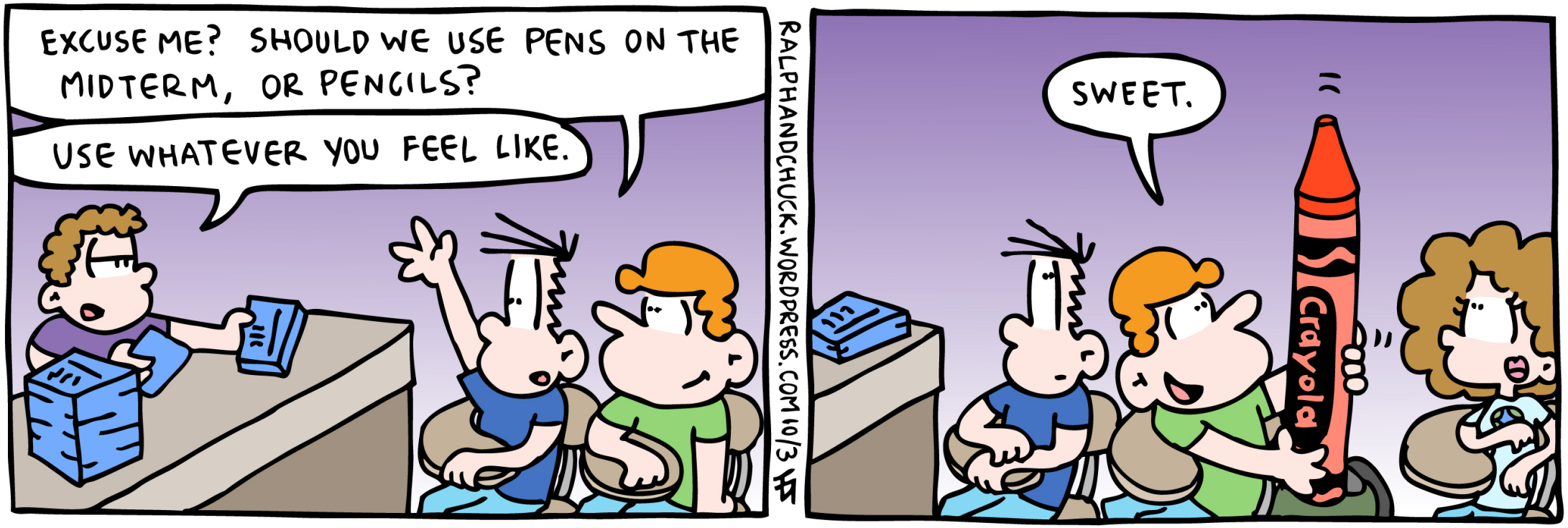
---

6. **Next lecture (Wedn)**
   - **Chapter 3** (3.1, 3.2) **+ 3.3 Loops**

Savitch Chapter 3 examples

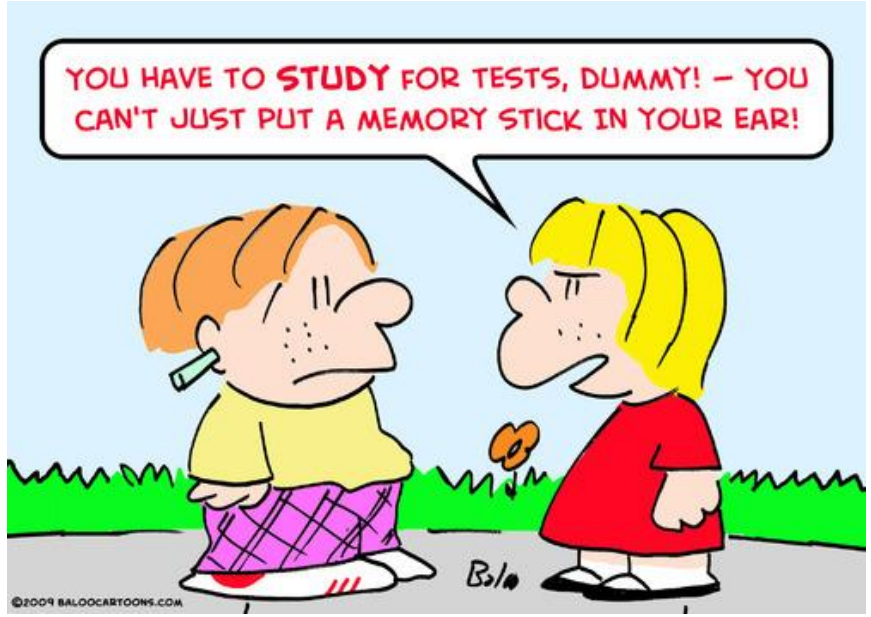
MIDTERM is Wed, Apr 18^{h} : 2 hours

- **See Canvas for content**


EXCUSE ME? SHOULD WE USE PENS ON THE MIDTERM, OR PENCILS?
USE WHATEVER YOU FEEL LIKE.
RALPHANDCHUCK.WORDPRESS.COM 10/3
SWEET.
Crayola


TEST


IF YOU DO NOT STUDY
YOU SHALL NOT PASS!


YOU HAVE TO STUDY FOR TESTS, DUMMY! – YOU CAN'T JUST PUT A MEMORY STICK IN YOUR EAR!
Balo
©2009 BALOOCARTOONS.COM

Homework and FoR feedback

HW 1 feedback

Average Score:	28.59
High Score:	30
Low Score:	21.5
Median:	29.5
Total:	43 submissions

Problem 2:

- About half interpreted the problem "change" method as convert 142 cents to 1 dollar, 1 quarter, 1 dime, 1 nickel and 2 pennies. [**the right answer**]
- The other half interpreted as 142 cent is 1 dollar, 5 quarters, 14 dimes, 28 nickels, 142 cents.
- It was graded so that either way was fine for interpretation because after reviewing the assignment statement, either way could work. However, **the assignment asked for an exact amount**. Therefore, people who gave the SECOND interpretation, lost some points (-0.5).
  - If they **used a (double) to give an exact answer** then I wouldn't reduce points.

FoR 3

1. **Switch statement:** How do we use them and when do we use them? Also, break, return, and default. What are they and do we need them?

2. **Scanner:** In general, most were confuse with the parsing/delimiter i.e. .next(), .nextDouble(), .nextLine() etc.

3. **If-statement and conditions:** the && and || and truth tables.

Few questions on multiway if-statement (else-if statement) and nested if-statements. Questions like when should we use them and how?

The Class `Scanner`

Console Input Using the `Scanner` Class

- Java includes a class for doing simple keyboard input named the `Scanner` class
- In order to use the `Scanner` class, a program must include the following line near the start of the file:

```
import java.util.Scanner
```

- This statement tells Java to
  - Make the `Scanner` class available to the program
  - Find the `Scanner` class in a library of classes (i.e., Java *package*) named `java.util`

Console Input Using the `Scanner` Class

- The following line creates an **object** of the class `Scanner` and names the object `keyboard` :

```
Scanner keyboard = new Scanner(System.in);
```

- Although a name like `keyboard` is often used, a `Scanner` object can be given any name
  - For example, in the following code the `Scanner` object is named `scannerObject`

```
Scanner scannerObject = new Scanner(System.in);
```

- Once a `Scanner` object has been created, a program can then use that object to perform keyboard input using methods of the `Scanner` class

Console Input Using the `Scanner` Class

- The method `nextInt` reads one `int` value typed in at the keyboard and assigns it to a variable:
  - `int numberOfPods = keyboard.nextInt();`
- The method `nextDouble` reads one `double` value typed in at the keyboard and assigns it to a variable:
  - `double d1 = keyboard.nextDouble();`
- Multiple inputs must be separated by *whitespace* and **read by multiple invocations** of the appropriate method
  - Whitespace is any string of characters, such as blank spaces, tabs, and line breaks that print out as white space

These are Delimiters

Console Input Using the `Scanner` Class

- The method `next` reads **one string of non-whitespace** characters **delimited by** whitespace characters such as **blanks** or the **beginning** or **end** of a line
- Given the code

```
String word1 = keyboard.next();
String word2 = keyboard.next();
```

and for the input line, what would be the value of word1

```
jelly beans
```

The value of `word1` would be `jelly`, and the value of `word2` would be `beans`

Console Input Using the `Scanner` Class

- The method `nextLine` reads an entire line of keyboard input
- The code,

```
String line = keyboard.nextLine();
```

- reads in an entire line and places the string that is read into the variable `line`
- The end of an input line is indicated by the **escape sequence** `'\n'`
  - This is the character input when the **Enter** key is pressed
  - On the screen it is indicated by the ending of one line and the beginning of the next line
- When `nextLine` reads a line of text, it reads the `'\n'` character, so the next reading of input begins on the next line
  - However, the `'\n'` does not become part of the string value returned (e.g., the string named by the variable `line` above does not end with the `'\n'` character)

Keyboard Input Demonstration (Part 1 of 2)

Display 2.6 Keyboard Input Demonstration

```
import java.util.Scanner;

public class ScannerDemo
{
   public static void main(String[] args)
   {
      Scanner keyboard = new Scanner(System.in);

      System.out.println("Enter the number of pods followed by");
      System.out.println("the number of peas in a pod:");
      int numberOfPods = keyboard.nextInt();
      int peasPerPod = keyboard.nextInt();

      int totalNumberOfPeas = numberOfPods*peasPerPod;

      System.out.print(numberOfPods + " pods and ");
      System.out.println(peasPerPod + " peas per pod.");
      System.out.println("The total number of peas = "
                                          + totalNumberOfPeas);
   }
}
```

Makes the `Scanner` *class available to your program.*

Creates an object of the class `Scanner` *and names the object* `keyboard`*.*

Each reads one `int` *from the keyboard*

(continued)

Savitch p.77

Keyboard Input Demonstration (Part 2 of 2)

Display 2.6 Keyboard Input Demonstration

SAMPLE DIALOGUE 1

```
Enter the number of pods followed by
the number of peas in a pod:
22 10
22 pods and 10 peas per pod.
The total number of peas = 220
```

The numbers that are input must be separated by whitespace, such as one or more blanks.

SAMPLE DIALOGUE 2

```
Enter the number of pods followed by
the number of peas in a pod:
22
10
22 pods and 10 peas per pod.
The total number of peas = 220
```

A line break is also considered whitespace and can be used to separate the numbers typed in at the keyboard.

Another Keyboard Input Demonstration (Part 1 of 3)

Display 2.7 Another Keyboard Input Demonstration

```
import java.util.Scanner;

public class ScannerDemo2
{
    public static void main(String[] args)
    {
        int n1, n2;
        Scanner scannerObject = new Scanner(System.in);

        System.out.println("Enter two whole numbers");
        System.out.println("seperated by one or more spaces:");

        n1 = scannerObject.nextInt();
        n2 = scannerObject.nextInt();
        System.out.println("You entered " + n1 + " and " + n2);

        System.out.println("Next enter two numbers.");
        System.out.println("Decimal points are allowed.");
```

Creates an object of the class `Scanner` *and names the object* `scannerObject`.

Reads one `int` *from the keyboard.*

Why it says “from the keyboard”

(continued)

Another Keyboard Input Demonstration (Part 2 of 3)

Display 2.7 Another Keyboard Input Demonstration

```
        double d1, d2;
        d1 = scannerObject.nextDouble();
        d2 = scannerObject.nextDouble();
        System.out.println("You entered " + d1 + " and " + d2);

        System.out.println("Next enter two words:");

        String word1 = scannerObject.next();
        String word2 = scannerObject.next();
        System.out.println("You entered \"" +
                                word1 + "\" and \"" + word2 + "\"");

        String junk = scannerObject.nextLine(); //To get rid of '\n'

        System.out.println("Next enter a line of text:");
        String line = scannerObject.nextLine();
        System.out.println("You entered: \"" + line + "\"");
    }
}
```

Reads one `double` *from the keyboard.*

Reads one word from the keyboard.

This line is explained in the Pitfall section "Dealing with the Line Terminator, `'\n'`*"*

Reads an entire line.

(continued)

Another Keyboard Input Demonstration (Part 3 of 3)

Display 2.7 Another Keyboard Input Demonstration

SAMPLE DIALOGUE

```
Enter two whole numbers
separated by one or more spaces:
  42   43
You entered 42 and 43
Next enter two numbers.
A decimal point is OK.
 9.99  57
You entered 9.99 and 57.0
Next enter two words:
jelly beans
You entered "jelly" and "beans"
Next enter a line of text:
Java flavored jelly beans are my favorite.
You entered "Java flavored jelly beans are my favorite."
```

Always “echo” (i.e. output) the user input for verification

Pitfall: Dealing with the Line Terminator, `'\n'`

- The method `nextLine` of the class `Scanner` reads **the remainder of a line of text starting wherever the last keyboard reading left off**
- This can cause problems when combining it with different methods for reading from the keyboard such as `nextInt`
- Given the code,

```
Scanner keyboard = new Scanner(System.in);
int n = keyboard.nextInt();
String s1 = keyboard.nextLine();
String s2 = keyboard.nextLine();
```

and the input,

```
2
Heads are better than
1 head.
```

what are the values of `n`, `s1`, and `s2`?

Pitfall: Dealing with the Line Terminator, `'\n'`

- Given the code and input on the previous slide

  `n` will be equal to `"2"`,

  `s1` will be equal to `""`, and

  `s2` will be equal to `"heads are better than"`

- If the following results were desired instead

  `n` equal to `"2"`,

  `s1` equal to `"heads are better than"`, and

  `s2` equal to `"1 head"`

then an extra invocation of `nextLine` would be needed to get rid of the end of line character (`'\n'`)

Methods in the Class `Scanner` (Part 1 of 3)

Display 2.8 Methods of the Scanner Class

The `Scanner` class can be used to obtain input from files as well as from the keyboard. However, here we are assuming it is being used only for input from the keyboard.

To set things up for keyboard input, you need the following at the beginning of the file with the keyboard input code:

```
import java.util.Scanner;
```

You also need the following before the first keyboard input statement:

```
Scanner Scannner_Object_Name = new Scanner(System.in);
```

The *Scannner_Object_Name* can then be used with the following methods to read and return various types of data typed on the keyboard.

Values to be read should be separated by whitespace characters, such as blanks and/or new lines. When reading values, these whitespace characters are skipped. (It is possible to change the separators from whitespace to something else, but whitespace is the default and is what we will use.)

Scannner_Object_Name`.nextInt()`

Returns the next value of type `int` that is typed on the keyboard.

(continued)

Methods in the Class `Scanner` (Part 2 of 3)

Display 2.8 Methods of the Scanner Class

Scannner_Object_Name`.nextLong()`

Returns the next value of type `long` that is typed on the keyboard.

Scannner_Object_Name`.nextByte()`

Returns the next value of type `byte` that is typed on the keyboard.

Scannner_Object_Name`.nextShort()`

Returns the next value of type `short` that is typed on the keyboard.

Scannner_Object_Name`.nextDouble()`

Returns the next value of type `double` that is typed on the keyboard.

Scannner_Object_Name`.nextFloat()`

Returns the next value of type `float` that is typed on the keyboard.

(continued)

Methods in the Class `Scanner` (Part 3 of 3)

Display 2.8 Methods of the Scanner Class

Scannner_Object_Name`.next()`

Returns the `String` value consisting of the next keyboard characters up to, but not including, the first delimiter character. The default delimiters are whitespace characters.

Scannner_Object_Name`.nextBoolean()`

Returns the next value of type `boolean` that is typed on the keyboard. The values of `true` and `false` are entered as the strings `"true"` and `"false"`. Any combination of upper- and/or lowercase letters is allowed in spelling `"true"` and `"false"`.

Scanner_Object_Name`.nextLine()`

Reads the rest of the current keyboard input line and returns the characters read as a value of type `String`. Note that the line terminator `'\n'` is read and discarded; it is not included in the string returned.

Scanner_Object_Name`.useDelimiter(`*New_Delimiter*`);`

Changes the delimiter for keyboard input with *Scanner_Object_Name*. The *New_Delimiter* is a value of type String. After this statement is executed, *New_Delimiter* is the only delimiter that separates words or numbers. See the subsection "Other Input Delimiters" for details.

Programming Tip: Prompt for Input

- A program should **always prompt the user** when he or she needs to input some data:

```
System.out.println(
  "Enter the number of pods followed by");
System.out.println(
  "the number of peas in a pod:");
```

Programming Tip: Echo Input

- **Always echo all input** that a program receives from the keyboard
- In this way a user can check that he or she has entered the input correctly
  - Even though the input is automatically displayed as the user enters it, echoing the input may expose subtle errors (such as entering the letter `"O"` instead of a zero)

Self-Service Checkout Line (Part 1 of 2)

Display 2.9 Self-Service Check Out Line

```
import java.util.Scanner;

public class SelfService
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);

        System.out.println("Enter number of items purchased");
        System.out.println("followed by the cost of one item.");
        System.out.println("Do not use a dollar sign.");

        int count = keyboard.nextInt();
        double price = keyboard.nextDouble();
        double total = count*price;

        System.out.printf("%d items at $%.2f each.%n", count, price);
        System.out.printf("Total amount due $%.2f.%n", total);

        System.out.println("Please take your merchandise.");
        System.out.printf("Place $%.2f in an envelope %n", total);
        System.out.println("and slide it under the office door.");
        System.out.println("Thank you for using the self-service line.");
    }
}
```

The dot after `%.2f` is a period in the text, not part of the format specifier.

(continued)

Write down the output, assume input, i.e. count of 10 items, $19.99 each (you have 2 minutes)

Self-Service Checkout Line (Part 2 of 2)

Display 2.9 Self-Service Check Out Line

SAMPLE DIALOGUE

```
Enter number of items purchased
followed by the cost of one item.
Do not use a dollar sign.
  10   19.99
10 items at $19.99 each.
Total amount due $199.90.
Please take your merchandise.
Place $199.90 in an envelope
and slide it under the office door.
Thank you for using the self-service line.
```

The Empty String

- A string can have any number of characters, including zero characters

  `""` is the empty string

- When a program executes the **`nextLine`** method to read a line of text, and the user types nothing on the line but presses the **Enter** key, then the **`nextLine`** Method reads the empty string

Other Input Delimiters

- The delimiters that separate keyboard input can be changed when using the `Scanner` class
- For example, the following code could be used to create a `Scanner` object and change the delimiter from whitespace to `"##"`

```
Scanner keyboard2 = new Scanner(System.in);
Keyboard2.useDelimiter("##");
```

- After invocation of the `useDelimiter` method, `"##"` and not whitespace will be the only input delimiter for the input object `keyboard2`

Changing the Input Delimiter (Part 1 of 3)

Display 2.10 Changing the Input Delimiter

```
import java.util.Scanner;

public class DelimiterDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard1 = new Scanner(System.in);
        Scanner keyboard2 = new Scanner(System.in);
        keyboard2.useDelimiter("##");
        //Delimiter for keyboard1 is whitespace.
        //Delimiter for keyboard2 is ##.
```

(continued)

Changing the Input Delimiter (Part 2 of 3)

Display 2.10 Changing the Input Delimiter

```
        String word1, word2;
        System.out.println("Enter a line of text:");
        word1 = keyboard1.next();
        word2 = keyboard1.next();
        System.out.println("For keyboard1 the two words read are:");
        System.out.println(word1);
        System.out.println(word2);
        String junk = keyboard1.nextLine(); //To get rid of rest of line.

        System.out.println("Reenter the same line of text:");
        word1 = keyboard2.next();
        word2 = keyboard2.next();
        System.out.println("For keyboard2 the two words read are:");
        System.out.println(word1);
        System.out.println(word2);
    }
}
```

(continued)

Changing the Input Delimiter (Part 3 of 3)

Display 2.10 Changing the Input Delimiter

SAMPLE DIALOGUE

```
Enter a line of text:
one two##three##
For keyboard1 the two words read are:
one
two##three##
Reenter the same line of text:
one two##three##
For keyboard2 the two words read are:
one two
three
```

Introduction to

File Input/Output

Introduction to File Input/Output

- The Scanner class can also be used to read from files on the disk
- Here we only present the basic structure of reading from text files
  - Some keywords are introduced without full explanation
  - More detail in Chapter 10
  - By covering the basics here your programs can work with real-world data that would otherwise be too much work to type into your program every time it is run

Text Input

- Import the necessary classes in addition to `Scanner`

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
```

- Open the file inside a `try`/`catch` block
  - If an error occurs while trying to open the file then execution jumps to the catch block
  - This is discussed in more detail in Chapter 9
- Use `nextInt()`, `nextLine()`, etc. to read from the Scanner like reading from the console, except the input comes from the file

Try/Catch Block

```
Scanner fileIn = null ; // initializes fileIn to empty
try
{
        // Attempt to open the file
        fileIn = new Scanner( new FileInputStream("PathToFile"));
}
catch (FileNotFoundException e)
{
        // If the file could not be found, this code is executed
        // and then the program exits
        System.out.println("File not found.");
        System.exit(0);
}
... Code continues here
```

Text File to Read

Display 2.11 Sample Text File, `player.txt`, to Store a Player's High Score and Name

```
100510
Gordon Freeman
```

This file should be stored in the same folder as the Java program in the following display

Program to Read a Text File

Display 2.12 Program to Read the Text File in Display 2.11

```
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class TextFileDemo
{
  public static void main(String[] args)
   {
     Scanner fileIn = null;  // Initializes fileIn to empty
     try
     {
       // Attempt to open the file
       fileIn = new Scanner(
           new FileInputStream("player.txt"));
     }
     catch (FileNotFoundException e)
     {
       // This block executed if the file is not found
       // and then the program exits
       System.out.println("File not found.");
       System.exit(0);
     }
```

try and catch is explained in more detail in Chapter 9.

The file `player.txt` *should be in the same directory as the Java program. You can also supply a full pathname to the file.*

Program to Read a Text File

```
        // If the program gets here then
        // the file was opened successfully
        int highscore;
        String name;

        System.out.println("Text left to read? " +
            fileIn.hasNextLine());
        highscore = fileIn.nextInt();
        fileIn.nextLine(); // Read newline left from nextInt()
        name = fileIn.nextLine();

        System.out.println("Name: " + name);
        System.out.println("High score: " + highscore);
        System.out.println("Text left to read? " +
            fileIn.hasNextLine());
        fileIn.close();
    }
}
```

This line is explained earlier in this chapter in the Pitfall section "Dealing with the Line Terminator '\n'"

Sample Dialogue

```
Text left to read? true
Name: Gordon Freeman
High score: 100510
Text left to read? False
```

Flow of Control

Fundamental Building Blocks of Programs

DATA

- **variables**

  Memory location/container

- **types**

  Sort of data

INSTRUCTIONS

- **control structures**

  Loops and branches ⇦ Start from

- **methods**

OOP *provides modular structure and tools to deal with complexity*

(versus a Big/Long program)

Flow of Control

As in most programming languages, *flow of control* in Java refers to its ***branching*** and ***looping*** mechanisms

- Java has several branching mechanisms:

  `if-else`, `if`, and **`switch`** statements

- Java has **three types of loop** statements:

  `while`, `do-while`, and **`for`** statements

- Most branching and looping statements are controlled by **Boolean expressions**
  - A Boolean expression evaluates to either **`true`** or **`false`**
  - The primitive type **`boolean`** may only take the values **`true`** or **`false`**

Branching with an `if-else` Statement

An `if-else` statement chooses between two alternative statements based on the value of a Boolean expression

```
if (Boolean_Expression)
   Yes_Statement;
else
   No_Statement;
```

Write down 2 examples

- The `Boolean_Expression` **must be enclosed in parentheses**
- If the `Boolean_Expression` is `true`, then the `Yes_Statement` is executed
- If the `Boolean_Expression` is `false`, then the `No_Statement` is executed

Compound Statements

- Each `Yes_Statement` and `No_Statement` can be a made up of a single statement or many statements
- ***Compound Statement***: A branch statement that is made up of a list of statements
  - A compound statement **must always be enclosed in a pair of braces** `{ }`
  - A compound statement can be used anywhere that a single statement can be used

Compound Statements

```
if (myScore > your Score)
{
   System.out.println("I win!");
   wager = wager + 100;
}
else
{
   System.out.println("I wish these were golf scores.");
   wager = 0;
}
```

Compound Statements

```
if (myScore > your Score)
{
    System.out.println("I win!");
    wager = wager + 100;
}
else
{
    System.out.println("I wish these were golf scores.");
    wager = 0;
}
```

Omitting the `else` Part

- The `else` part may be omitted to obtain what is often called an `if` statement

```
if (Boolean_Expression)
        Action_Statement;
```

  - If the `Boolean_Expression` is true, then the `Action_Statement` is executed
  - The `Action_Statement` can be a single or compound statement
  - **Otherwise**, nothing happens, and the program goes on to the next statement

```
if (weight > ideal)
    calorieIntake = calorieIntake - 500;
```

Nested Statements

- **`if-else`** statements and **`if`** statements both contain smaller statements within them
  - For example, single or compound statements
- In fact, any statement at all can be used as a subpart of an **`if-else`** or **`if`** statement, including another **`if-else`** or **`if`** statement
  - Each level of a nested **`if-else`** or **`if`** **should be indented further** than the previous level
  - Exception: *multiway* **`if-else`** statements

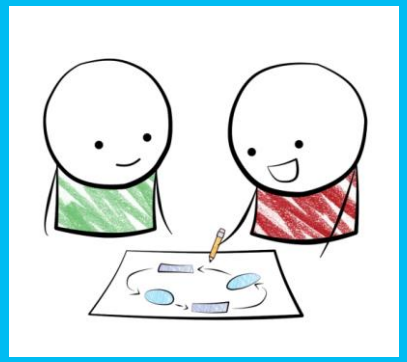
Multiway `if-else` Statements

- The multiway **`if-else`** statement is simply a normal **`if-else`** statement that **nests** another **`if-else`** statement at every **`else`** branch
  - It is indented differently from other nested statements
  - All of the **`Boolean_Expressions`** are aligned with one another, and their corresponding actions are also aligned with one another
  - The **`Boolean_Expressions`** are evaluated in order **until one that evaluates to `true`** is found
  - The final **`else`** is optional

Multiway `if-else` Statement

```
if (Boolean_Expression)
   Statement_1
else if (Boolean_Expression)
   Statement_2
     ……….
else if (Boolean_Expression_n)
   Statement_n
else
  Statement_For_All_Other_Possibilities
```

Hands on Activity 2



Work in pairs