SWEN20003
Object Oriented Software Development

Input and Output

Semester 1, 2019

Assess Yourself

During a game of Age of Empires, one of your soldiers is wounded by an enemy, and needs to be healed by a medic.

How would you implement this behaviour?

How would you *model* this situation?

Assess Yourself

```
public class Soldier {

    public static final double MAX_HEALTH = 100;

    public double health = MAX_HEALTH;

    public void doDamage(double damage) {
        this.health -= damage;
        if (health <= 0) {
            this.die();
        }
    }

    public String toString() {
        return String.format("Soldier at %f health", health);
    }
}
```

Assess Yourself

```
public class Medic {

    public static final double HEAL_RATE = .002;

    public void heal(Soldier soldier, int delta) {
        double health = HEAL_RATE * delta;
        soldier.health += health;
    }
}
```

Assess Yourself

```
public class Program {
    public static void main(String args[]) {

        Soldier soldier = new Soldier();
        Medic medic = new Medic();
        System.out.println(soldier);

        soldier.damage(10);
        System.out.println(soldier);

        int delta = getTimeSinceLastFrame();

        medic.heal(soldier, delta);
        System.out.println(soldier);
    }
}
```

The Road So Far

- OOP Foundations
  - Classes and Objects
  - Strings and Wrappers
  - Formatting
  - Methods and Abstraction

Lecture Objectives

After this lecture you will be able to:

- Accept input to your programs through:
  - Command line arguments
  - User input
  - Files
- Write output from your programs through:
  - Standard output (terminal)
  - Files
- Use files to store and retrieve data during program execution
- Manipulate data in files (i.e. for computation)

Input:

Command Line Arguments

Command Line Arguments

Let's take a look back at "Hello World"

```
public static void main(String[] args)
```

What exactly is this?

Command Line Arguments

```
void main(String[] args)
```

- `args` is a variable that stores command line arguments
- `String[]` means that `args` is an *array* of Strings
- We'll cover arrays in detail next lecture, but we'll look at some basics in a moment
- But first...

Entering Arguments - Terminal

- If you compile and run Java from the terminal the syntax is very similar to C

```
java MyProg Hello World 10
```

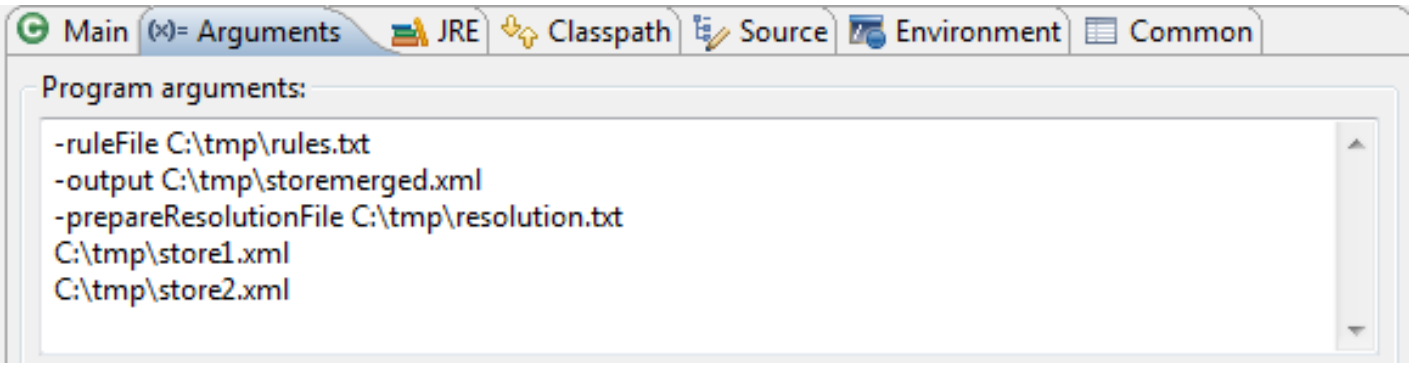
- This fills the `args` variable with three elements, `"Hello"`, `"World"` and `"10"`
- For multiword Strings, remember to use quotes
- Also note that `"10"` is a `String`, not an `int`

```
java MyProg "Hello World" 10
```

- This fills the `args` variable with two elements, `"Hello World"` and `"10"`

Entering Arguments - IDE

- Because IDEs do a lot of "behind the scenes" magic, command line arguments are a bit different
- In Eclipse we have to set the "run configuration" (Run → Run Configurations...) to provide command line arguments
- Here's an **example**



- Other IDEs will have similar settings; a good time to practice your Google skills!

What Next?

- How do you actually use the arguments once they are put into the program?
- Access the elements of the array by *indexing*
- Identical syntax to accessing array elements in C, or list/tuple elements in Python

```
java MyProg "An" "Argument" "This is another argument"
```

```
System.out.println(args[0]);
System.out.println(args[1]);
System.out.println(args[2]);
```

```
"An"
"Argument"
"This is another argument"
```

Keyword

Command Line Argument: Information or data provided to a program when it is *executed*, accessible through the `args` variable.

Assess Yourself

Write a program that creates a `Person` object from three **command line arguments**, and then outputs the object as a `String`.

A `Person` is created from three arguments:

- `int age` - age, in years
- `double height` - height, in metres
- `String name` - name, as a `String`

Assess Yourself

```java
public class Program {
    public static void main(String[] args) {
        int age = Integer.parseInt(args[0]);
        double height = Double.parseDouble(args[1]);
        String name = args[2];

        Person person = new Person(age, height, name);
        System.out.println(person);
    }
}
```

Example input:

```
java Program 27 1.68 "Matt De Bono"
```

Example output:

```
"Matt De Bono - age: 27, height: 168cm"
```

Assess Yourself

- No interactivity
- Usually for program configuration
- Only when the question tells you! Probably never.
- Let's look at the interactive alternative

Input:

Scanner

Scanner

- Java offers a much more powerful approach to input than C and Python called the `Scanner`
- We'll look at some of the capabilities, but check out the full documentation here

Scanner

- Need to import the library first

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

- Then we create the `Scanner`

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

- Only ever create **one** `Scanner` for each program, or bad things happen

Creating a Scanner

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

- The stream/pipe to receive data from, in this case *standard input* (the terminal)

Keyword

System.in: An object representing the *standard input* stream, or the command line/terminal.

Using a Scanner

- Once we've created the `Scanner`, what do we do with it?
- `Scanner` has a number of methods used to read data
- The obvious first:

```
String s = scanner.nextLine();
```

- Reads a single line of text, up until a "return" or newline character

Using a Scanner

- But there's more:

```
boolean b = scanner.nextBoolean();
int i = scanner.nextInt();
double d = scanner.nextDouble();
```

- Reads a single value that matches the method name (`boolean`, `int`, etc...)

Assess Yourself

```java
import java.util.Scanner;

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

        System.out.println("Enter your input: ");
        double d = scanner.nextDouble();
        String s1 = scanner.next();
        String s2 = scanner.nextLine();

        System.out.format("%3.2f,%s,%s", d, s2, s1);
    }
}
```

Input: `5.2  Hello,World Are there any more words?`

Output: `5.20, Are there any more words?,Hello,World`

Assess Yourself

```java
import java.util.Scanner;

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

        System.out.println("Enter your input: ");
        double d = scanner.nextDouble();
        float f = scanner.nextFloat();
        int i = scanner.nextInt();

        System.out.format("%3.2f , %3.2f , %3d", d, f, i);
    }
}
```

Input: 5 6.7 7.2

Output: Error

Pitfall: nextXXX

- Scanner does not automatically downcast (i.e. float to int)
- When using `nextXXX`, be sure that the input matches what is expected by your code!

Assess Yourself

```java
import java.util.Scanner;

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

        System.out.println("Enter your input: ");
        double d = scanner.nextDouble();
        String s1 = scanner.nextLine();
        String s2 = scanner.nextLine();

        System.out.format("%3.2f , %s , %s", d, s1, s2);
    }
}
```

Input: 5
6.7
7.2

Output: 5.00 , , 6.7

Pitfall: Mixing nextXXX with nextLine

- `nextLine` is the **only** method that "eats" newline characters
- In some cases, you may have to follow `nextXXX` with `nextLine`, if your input is on multiple lines

Other Features

```
scanner.hasNext()
scanner.hasNextXXX()
```

Keyword

.hasNext: Returns `true` if there is *any* input to be read

Keyword

.hasNextXXX: Returns `true` if the next "token" matches *XXX*

Assess Yourself

Write a program that accepts three **user inputs**, creates an IMDB entry for an `Actor`, and prints the object:

- `String name` - the name of a character in a movie/TV show
- `double rating` - a rating for that character
- `String review` - a review of that character

Here is an example of the output format:

```
"You gave Tony Stark a rating of 9.20/10"
"Your review: 'I wish I was like Tony Stark...'"
```

Assess Yourself

```
public class Actor {
    public static final int MAX_RATING = 10;

    public String name;
    public double rating;
    public String review;

    public Actor(String name, double rating, String review) {
        this.name = name;
        this.rating = rating;
        this.review = review;
    }

    public String toString() {
        return String.format("You gave %s a rating of %f/%d\n"
                    name, rating, MAX_RATING)
                + String.format("Your review: '%s'", review);
    }
}
```

Assess Yourself

```java
import java.util.Scanner;

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

        String name = scanner.nextLine();

        double rating = scanner.nextDouble();
        scanner.nextLine();

        String comment = scanner.nextLine();

        Actor actor = new Actor(name, rating, comment);
        System.out.println(actor);
    }
}
```

Input:

Reading Files

Reading Files

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class Program {
    public static void main(String[] args) {

        try (BufferedReader br =
            new BufferedReader(new FileReader("test.txt"))) {

            String text;

            while ((text = br.readLine()) != null) {
                <something useful>
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

Reading Files - Classes

```
try (BufferedReader br = new BufferedReader(new FileReader("test.txt")))
```

- Creates two objects:
  - FileReader - A low level file (`"test.txt"`) for simple character reading
  - BufferedReader - A higher level file that permits reading `Strings`, not just characters
- `try` will automatically close the file once we're done
- `br` is our file variable

Reading Files - Methods

```
while ((text = br.readLine()) != null)
```

- `br.readLine()`: Reads a single line from the file
- `text =`: Assigns that line of text to a variable
- `!= null`: Then check if anything was actually read

Reading Files - Errors

```
catch (IOException e) {
    e.printStackTrace();
}
```

- `catch` - Acts as a safeguard to potential errors, prints an error message if anything goes wrong; more on Exceptions later

Reading Files - Libraries

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
```

- All the classes that make the example go; these make file input possible

Reading Files - Scanner

```
try (Scanner file = new Scanner(new FileReader("test.txt"))) {

    while (scanner.hasNextLine()) {
    <something useful>
    }

} catch (Exception e) {
    e.printStackTrace();
}
```

- Works the same as `BufferedReader`, but allows us to *parse* the text, as well as read it
- Smaller buffer size (internal memory), slower, works on smaller files

Assess Yourself

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class Program {
    public static void main(String[] args) {

        try (BufferedReader br =
            new BufferedReader(new FileReader("test.txt"))) {
            String text;
            int count = 0;

            while ((text = br.readLine()) != null) {
                String words[] = text.split(" ");

                count += words.length;
            }

            System.out.println("# Words = " + count);
        } catch (Exception e) {
            e.printStackTrace();
        }

        }
}
```

Assess Yourself

```java
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class Program {
    public static void main(String[] args) {

        try (BufferedReader br =
            new BufferedReader(new FileReader("test.html"))) {

            String text;

            int count = 0;

            while ((text = br.readLine()) != null) {
                count = text.contains("<h1>") ? count + 1 : count;
            }

            System.out.println("# Headers: " + count);
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

Assess Yourself

```java
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.Random;

public class Program {
    public static void main(String[] args) {
        final int MAX_NUM = 10000;
        final int ITERATIONS = 1000000;

        Random rand = new Random();

        try (PrintWriter pw =
            new PrintWriter(new FileWriter("test.txt"))) {

            int nums[] = new int[MAX_NUM];

            for (int i = 0; i < ITERATIONS; i++) {
                nums[rand.nextInt(MAX_NUM)] += 1;
            }
            for (int i = 0; i < nums.length; i++) {
                pw.format("%4d: %4d\n", i, nums[i]);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Assess Yourself

```java
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        try (PrintWriter pw =
            new PrintWriter(new FileWriter("test.html"))) {

            pw.println("<h1>The Chronicles of SWEN20003</h1>");

            while (scanner.hasNext()) {}
                String text = scanner.nextLine();

                pw.println("<p>" + text + "</p>");
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Assess Yourself

```java
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class Program {
    public static void main(String[] args) {

        try (BufferedReader br =
            new BufferedReader(new FileReader("recipe.csv"))) {
            String text;
            int count = 0;

            while ((text = br.readLine()) != null) {
                String cells[] = text.split(",");

                String ingredient = cells[0];
                double cost = Double.parseDouble(cells[1]);
                int quantity = Integer.parseInt(cells[2]);

                System.out.format("%d %s will cost $%.2f\n", quantity,
                ingredient, cost*quantity);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

Reading CSV files

- CSV = *Comma Separated Value*
- Somewhat equivalent to a spreadsheet
- Usually contains a header row to explain columns
- Example:

```
Ingredient,Cost,Quantity
Bananas,9.2,4
Eggs,1,6
```

- **Required knowledge for Projects!**

Output:

Writing Files

File Output

```
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;

public class Program {
    public static void main(String[] args) {
        try (PrintWriter pw =
            new PrintWriter(new FileWriter("test.txt"))) {

            pw.println("Hello World");
            pw.format("My least favourite device is %s and its price is $%d",
                "iPhone", 100000);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

File Output - Classes

```
try (PrintWriter pw = new PrintWriter(new FileWriter("test.txt")))
```

- Creates two objects:
  - `FileWriter` - A low level file (`"test.txt"`) for simple character output, used to create...
  - `PrintWriter` - A higher level file that allows more sophisticated formatting (same methods as `System.out`)
- `try` will automatically close the file once we're done
- `pw` is our file variable

File Output - Methods

```
pw.print("Hello ");
pw.println("World");
pw.format("My least favourite device is %s and its price is $%d",
    "iPhone", 100000);
```

- `pw.print` - Outputs a String
- `pw.println` - Outputs a String with a new line
- `pw.format` - Outputs a String, and allows for format specifiers

File Output - Errors

```
catch (IOException e) {
    e.printStackTrace();
}
```

- `catch` - Acts as a safeguard to potential errors, prints an error message if anything goes wrong; more on `Exceptions` later

File Output - Libraries

```
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;
```

- All the classes that make the example go; these make file output possible

File Input and Output

- You will **not** be expected to write all of this from memory in the test/exam
- **If** you are asked to manipulate files, you will be given sufficient scaffold/supporting methods
- For now, all you need to do is practice, and understand; we'll talk about assessment closer to the test

Assess Yourself

Implement a rudimentary survey/voting system, by writing a program that continuously expects a single input from the user. This input will be one of three options, in response to the question "Which is your favourite Star Wars trilogy?"

The valid responses are `0` (for the "Original" trilogy), `1` ("New"), and `2` ("The other one").

Once the input has ended, your program should output the results of the survey, one option per line, as below.

Execution:

```
0
0
1
1
1
2
Original Trilogy: 2
New Trilogy: 3
Other Trilogy: 1
```

Assess Yourself

```java
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {

        final int N_OPTIONS = 3;

        final int ORIGINAL = 0;
        final int NEW = 1;
        final int OTHER = 2;

        int results[] = new int[N_OPTIONS];

        Scanner scanner = new Scanner(System.in);

        while (scanner.hasNextInt()) {
            int vote = scanner.nextInt();
            results[vote] += 1;
        }

        System.out.println("Original Trilogy: " + results[ORIGINAL]);
        System.out.println("New Trilogy: " + results[NEW]);
        System.out.println("Other Trilogy: " + results[OTHER]);

    }
}
```

Assess Yourself

Follow up:

- What would you do if there were five valid inputs?
- What about `n` inputs?
- What about allowing the user to **tell you** the options, then getting votes?

Combining Reading and Writing

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.PrintWriter;

public class Program {
    public static void main(String[] args) {

        try (BufferedReader br = new BufferedReader(new FileReader("input.txt"));
            PrintWriter pw = new PrintWriter(new FileWriter("output.txt"))) {

            String text;

            while ((text = br.readLine()) != null) {
                <something really useful>
            }

        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

Application #1: Data Storage/Retrieval

```
/** Using files to store intermediate data during computation */

final int MAX_DATA = 1000;

try (BufferedReader br = new BufferedReader(new FileReader("input.txt"));
    PrintWriter pw = new PrintWriter(new FileWriter("output.txt", true))) {

    // Recover data from previous run
    String oldData[] = loadPreviousData(br);

    String newData[] = new String[MAX_DATA];

    int count = 0;

    while (magicalComputationNeedsDoing()) {
        newData[count] = magicalComputation(oldData);

        count += 1;

        // Once we do enough computation, store the results just in case
        if (count == MAX_DATA) {
            writeData(pr, newData);
            count = 0;
        }
    }

}
```

Application #2: Data Manipulation

```java
/** Using Java to parse/manipulate/convert/etc. files */

try (BufferedReader br = new BufferedReader(new FileReader("input.txt"));
    PrintWriter pw = new PrintWriter(new FileWriter("output.txt"))) {

    String text;

    while ((text = br.readLine()) != null) {
        // Manipulate the input file
        String newText = magicalComputation(text);

        // Write to output file
        pw.println(newText);
    }

}
```

Assess Yourself

1. Write a program that accepts a `filename` from the user, which holds the marks for students in SWEN20003. Your program must then process this data, and output a histogram of the results
2. Extend your program so that it accepts two more inputs for the `min` and `max` values for the data
3. Extend your program so that it accepts one more input for the `width` of each "bin" in the histogram

Assess Yourself

```
import java.util.Scanner;
import java.io.File;

public class Program {
    public static void main(String[] args) {

        System.out.print("Enter filename: ");
        String filename = scanner.nextLine();

        System.out.print("Enter min value: ");
        int min = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter max value: ");
        int max = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter bin width: ");
        int width = scanner.nextInt();
        scanner.nextLine();

        int data[] = new int[max-min + 1];

        int total = 0;

        ...

    }
}
```

Assess Yourself

```
try (Scanner file = new Scanner(new File(filename))) {
    // Skip the first line
    file.nextLine();

    while (file.hasNext()) {
        String line[] = file.nextLine().split(",");

        int d = Integer.parseInt(line[1]);

        data[d - min] += 1;
        total += 1;
    }

    ...

} catch (Exception e) {
    e.printStackTrace();
}
```

Assess Yourself

```
// Print out graph
for (int i = 0; i < data.length; i += width) {
    int sum = 0;

    // Bundle into *width* sized blocks
    for (int j = 0; j < width && i + j < data.length; j++) {
        sum += data[i+j];
    }

    int percentage = (int) (100 * (1.0 * sum)/total);
    String bar = "";

    if (percentage > 0) {
        bar = String.format("%" + percentage + "s", " ")
        .replace(" ", "=");
    }

    int lower = i + min;
    int upper = lower + width - 1;

    // Print the block
    System.out.format("%03d-%03d: %s\n", lower, upper, bar);
}
```

Metrics

Write a program that takes three inputs from the user:

- `String`, a unit of measurement
- `int`, the number of units
- `String`, an ingredient in a recipe

Your code should write in the following format to a file called `"recipe.txt"`:
`"- Add 300 grams of chicken"`

Bonus Task:
Open the file in "append" mode; this means the file will be added to, rather than overwritten, each time you run your code.

Metrics

Write a program that accepts a `filename` from the user, and then processes that file, recording the frequency with which **words** of different lengths appear.

Metrics

Write a program that accepts a HTML `filename` from the user, and then takes continuous user input and writes it to the file; essentially a Java based HTML writer.

Bonus #1: add validation to detect valid HTML tags (`<p>`, `<h1>`, etc.).

Bonus #2: add "shortcuts"; for example, entering `{text}` might make *text* automatically bold.