Lecture 13
Practical Programming

“Practical”?

- Real programs:
  - Manipulate files
  - Can be called with arguments
  - Are often event-based
- ...but we haven’t done any of that so far!
- Because there’s so much to learn, and so little time! :(

Graphical Programs

- We’re going to make a (bad) text editor!
- Using Swing!
  - Pros: used in the book, easy to get started
  - Cons: being replaced by JavaFX, looks terrible
- Principles are largely the same
- There will be **no** graphics programming on the exam!

Components

- Graphical programs consist of (many) components
- Inherit (unsurprisingly) from… `Component`
- In our program:
  - `JFrame`: the window
  - `JTextArea`: an area for text entry
  - `JButton`: a button
  - `JPanel`: used to lay out the other components

Let’s Get Started!

Event-Based Programming

- Most programs are *event-based*
- They sit around doing nothing until something happens – an *event*
  - A network request
  - A keystroke
  - A mouse click
  - ...
- Then they respond to the event, and go back to doing nothing

ActionListener

- Interface for classes that can *react to events*
- Has one important method:

```
void actionPerformed(ActionEvent event)
```

- Objects implementing `ActionListener` can be added to `Component`s
- `actionPerformed` is called whenever an event happens on that component

Adding a “Clear” Button

File Handling

- A text editor that can’t save or load files is pretty useless
- So far we’ve used `java Program < file.txt`
- But this is very inflexible
  - What if we want to read more than one file?
  - What if we don’t know which file to read when we start the program?

File Handling

- We can use the `File` class to work with files

```
File file = new File("my_file.txt");
if(file.exists()) {
  System.out.println("The file exists!");
  file.delete();
  System.out.println("Now it’s gone!");
} else {
  System.out.println("The file does not exist!");
}
```

- `File` lives in package `java.io`.
- https://docs.oracle.com/javase/7/docs/api/java/io/File.html

Reading Files

- We can construct a `Scanner` from a `File`

```
File file = new File("my_file.txt");
try {
  Scanner scan = new Scanner(file);
  while(scan.hasNextLine()) {
    System.out.println(scan.nextLine());
  }
  scan.close();
} catch (FileNotFoundException e) {
  System.out.println("The file does not exist!");
  System.exit(1);
}
```

Reading Files

- We can construct a `Scanner` from a `File`

```
File file = new File("my_file.txt");
try {
  Scanner scan = new Scanner(file);
  while(scan.hasNextLine()) {
    System.out.println(scan.nextLine());
  }
  scan.close();
} catch (FileNotFoundException e) {
  System.out.println("The file does not exist!");
  System.exit(1);
}
```

Don't forget to close the scanner; the file will be *locked* until we do!

`FileNotFoundException` is a *checked* exception: we *must* handle it, or our program won't compile!

Adding a “Save” Button

Writing Files

- We can construct a `FileWriter` from a `File`
- ...which we then use to construct a `PrintWriter`

```
File file = new File("my_file.txt");
try {
  FileWriter fileWriter = new FileWriter(file);
  PrintWriter writer = new PrintWriter(fileWriter);
  writer.println("Hello, I'm a line of text!");
  writer.println("And so am I!");
  writer.close();
} catch (IOException e) {
  System.out.println("Something went wrong!");
  System.exit(1);
}
```

- `FileWriter` and `PrintWriter` live in package `java.io`.

Writing Files

- We can construct a `FileWriter` from a `File`
- ...which we then use to construct a `PrintWriter`

```
File file = new File("my_file.txt");

FileWriter fileWriter = new FileWriter(file);
PrintWriter writer = new PrintWriter(fileWriter);
writer.println("Hello, I'm a line of text!");
writer.println("And so am I!");
writer.close();
} catch (IOException e) {
  System.out.println("Something went wrong!");
  System.exit(1);
}
```

The writer also needs to be closed, to avoid locking the file for longer than necessary

`IOException` is also checked, and could mean a disk error, trying to create a file in a directory that doesn't exist, etc.

- `FileWriter` and `PrintWriter` live in package `java.io`.

Adding a “Load” Button

Command Line Arguments

- Like methods, programs can take *arguments*
- `cp my_file.txt my_copy.txt`
  - Copies `my_file.txt` to `my_copy.txt`
- `wc -l my_file.txt`
  - Counts the number of lines in `my_file.txt`
- `cat a.txt b.txt c.txt > abc.txt`
  - Concatenates `a`, `b` and `c.txt`, and writes the result to `abc.txt`

Command Line Arguments

- When you double-click `file.txt`
  - The operating system figures out which program is used to open `.txt` files
  - The operating system calls that program with `file.txt` as its first *argument*
  - `C:\Windows\system32\notepad.exe file.txt` on a new Windows machine

Command Line Arguments

```
public class Program {
  public static void main(String[] args) {
    for(String arg: args) {
      System.out.println(arg);
    }
  }
}
```

Did you ever wonder what `String[] args` is?

Command Line Arguments

```
public class Program {
  public static void main(String[] args) {
    for(String arg: args) {
      System.out.println(arg);
    }
  }
}
```

Did you ever wonder what `String[] args` is?

```
java Program Hello, I am the arguments!
```

Prints:

```
Hello,
I
am
the
arguments!
```

Command Line Arguments

```
public class Program {
  public static void main(String[] args) {
    for(String arg: args) {
      System.out.println(arg);
    }
  }
}
```

Did you ever wonder what `String[] args` is?

```
java Program "Hello, I am the arguments!"
```

Prints:

```
Hello, I am the arguments!
```

Starting the Program with a File

Reading and Exercises

- Reading
  - 5.5, 16.6, 9.4
- Exercises
  - 5.7, exercises 5, 6, 8
  - Bonus exercise: Contact List (see course website)