Advanced Programming Techniques in Java

Class objectives

- Files I/O (Chapter 6)
- Arrays (Chapter 7)

Review: Compil

```
import java.io.*;      // for File
import java.util.*;    // for Scanner

public class ReadFile { public static vo
    main(String[] args) {
        Scanner input = new Scanner(new F
        String text = input.next();
        System.out.println(text);
    }
}
```

- The program fails to compile with the follo

```
ReadFile.java:6: unreported exception
must be caught or declared to be thro
        Scanner input = new Scanner(n
```

Exceptions

- An **exception** is an error that occurs at run
  "exceptional" circumstance
  - Dividing an integer by 0
  - Calling substring on a `String` and passing to
  - Trying to read the wrong type of value from a S
  - Trying to read a file that does not exist

`StringIndexOutOfBoundsException`
`IllegalArgumentException`

Review: Excepti

- *Checked* exceptions
  - normally not due to programmer error
  - generally, beyond the control of the programmer
  - all I/O errors are checked exceptions
  - eg. FileNotFoundException
- *Unchecked exceptions*
  - programmer error (try to prevent them with d
  - a serious external condition that is unrecovera
  - ArrayIndexOutOfBoundsException

Review: Excepti

- When using a `Scanner` to process a file, we c
  - `FileNotFoundException`:
    - If the file that we specify isn't there
    - If the file is inaccessible for some reason
- We say that a program with an error "throws" a
- It is also possible to "catch" (handle or fix) an ex
- The compiler checks that we either
  - **Declare that we don't handle it**
  - **Handle it (`try/catch`)**

- We do this by adding a throws clause

Token-based vs. processing

- Token-based: The practice of processing input time or one number at a time)
- Line-based: The practice of processing input lin of input at the time)

Input token

- A token is unit of user input, separated by whi

- The `Scanner` methods don't necess

of output

- If the input file contains the followin

```
23 3.12
"Iraklis"
```

- The `Scanner` can interpret the tokens as t

Token	Type(s)
`23`	`int`, `double`, `Str`
`3.12`	`double`, `String`
`"Iraklis"`	`String`

Files and input c

- Consider a file `weather.txt` that contains

```
16.2 23.5
19.1 7.4 22.8

18.5 -1.8 14.9
```

- A `Scanner` views all input as a stream of ch

```
16.2 23.5\n19.1 7.4 22.8\n\n18.5 -1.
^
```

Consuming toke

- **Consuming input** means reading input and adv

Calling `nextInt` etc. moves the cursor past the

```
16.2 23.5\n19.1 7.4 22.8\
1.8 14.9\n ^ double d = input.nextD
// 16.2 16.2 23.5\n19.1 7.4 22.8\n\
1.8 14.9\n
^
16.2 23.5\n19.1 7.4 22.8\n\n18.5 -1
```

- If you attempted to call `nextDouble` again
  `NoSuchElementException`

`Scanner` tests fo

Method	Descript
`hasNext()`	returns `true` if there is a

`hasNextInt()`	returns `true` if there is a and it can be read as an
`hasNextDouble()`	returns `true` if there is a and it can be read as a `d`

- These methods of the `Scanner` do not co
  information about what the next token will b
  - Useful to see what input is coming, and to avoid c
- They can be used with a console `Scanner`,

Files input: Ques

- Consider a file `weather.txt` that contains

```
16.2 23.5
19.1 7.4 22.8
```

```
18.5 -1.8 14.9
```

- Write a program that prints the change in ter
  neighboring days

```
16.2 to 23.5, change = 7.3
23.5 to 19.1, change = -4.4
19.1 to 7.4, change = -11.7
7.4 to 22.8, change = 15.4
22.8 to 18.5, change = -4.3
18.5 to -1.8, change = -20.3
-1.8 to 14.9, change = 16.7
```

Files input: Answ

```
// Displays changes in temperature from data

import java.io.*;    // for File import java.
public class Temperatures { public static voi
throws FileNotFoundException { Scanner input
File("weather.txt")); double prev = input.nex
(input.hasNextDouble()) { double next = input
        System.out.println(prev + " to " + ne
        prev = next;
    }
  }
}
```

Files input: C

- Modify the temperature program to handle fil
  tokens (by skipping them)

```
16.2 23.5 Tuesday 19.1  Wed 7.4
THURS.TEMP 22.8

18.5 -1.8 14.9
16.1
```

- You may assume that the file begins with a rea

Files input: Answ

```
// Displays changes in temperature from data in an input

import java.io.*;    // for File import java.util.*;  //
public class Temperatures2 { public static void main(Str
throws FileNotFoundException {
        Scanner input = new Scanner(new
        File("weather.txt")); double prev =
        input.nextDouble();   while (input.hasNext()) {
        (input.hasNextDouble()) { double next =
        input.nextDouble();
                System.out.println(prev + " to " + next
                prev = next;
            } else { input.next();  // throw away unwant
                token
            }
        }
    }
}
```

Line-based `Scan`

Method	Des
`nextLine()`	returns next entire line of
`hasNextLine()`	returns `true` if there are (always true for console inpu

```
Scanner input = new Scanner(new
while (input.hasNextLine()) {
    String line = input.nextLine
    processLine(line);
}
```

`Scanner` **on str**

- So far we have seen that you can pass to `Scan`
  and the object `File`
- We can also pass the object `String`

`Scanner`

stri

- A `Scanner` can tokenize the content of a

<u>Syntax</u>:

```
Scanner <name> = new Scanner(<Str
```

<u>Example</u>:

```
String text = "15  3.2 hello   9
27.5"; Scanner scan = new
Scanner(text);
```

```
int num = scan.nextInt();   //
15 double num2 =
scan.nextDouble();  //3.2 String word
= scan.next(); //hello
```

Mixing lines and

Input file `input.txt`:	Output t
The quick brown fox jumps over the lazy dog.	Line ha words Line ha words

```
// Counts the words on each line of a fil
Scanner input = new Scanner(new
File("input.txt")); while
(input.hasNextLine()) {
        String line = input.nextLine();
        Scanner lineScan = new Scanner(li

        // process the contents of this l
        int count = 0;
        while (lineScan.hasNext()) {
                String word =
                lineScan.next(); count++;
        }
        System.out.println("Line has " + 
        words");
}
```

File output

- So far we have sent the output of a program
  - `System.out.print`
  - `System.out.println`
- You can write output to a file:

Syntax

```
PrintStream <name> = new PrintStream (new Fi
```

Example

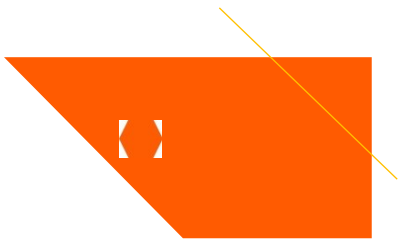
```
PrintStream output = new PrintStream(new Fi
output.println("Hello, file!");
output.println("This is a second line of ou
```

D
Pri

Syntax

```
PrintStream <name> = new PrintStream (new File
```

- If the given file does not exist, it is created
- If the given file already exists, it is overwritten
- The output you print appears in a file, not on t
  - You will have to open the file with an editor to see i
- Do not open the same file for both reading (`Sc`
  (`PrintStream`) at the same time



You will overwrite your input file with ar

Details about `Pri`

<u>Syntax</u>

```
PrintStream <name> = new PrintStream (new File
```

- This line of code can generate an exception if J
  - You might not have permission to write to the direc
  - You might be locked because another file is using it
- To handle the exception, you need to include t method contains this line of code or surround

```
System.out a
```

- The console output object `System.o`

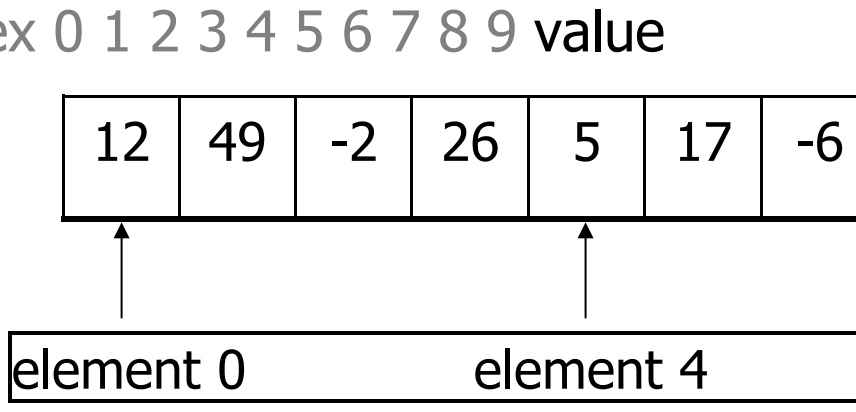
```
PrintStream out1 = System.out;
PrintStream out2 = new PrintStream(new Fi
out1.println("Hello, console!");   // goe
out2.println("Hello, file!");      // goe
```

- A reference to it can be stored in a `PrintSt`
- You can pass `System.out` to a method as

Arrays

Arrays

- An array is a collection (object) of data values
- An array can be thought as a sequence of box



- Each box contains one of the data values in th
- Each element has a numeric index. The first el

Declaring an

- We often declare and create an array in the sa

```
<type>[] <array_name> = new <type>[<le
```

type of the individual elements

```
int[] A = new int[10];
```

The length of an a

- The length of an array is the number of eleme
- The length of an array can be obtained as follo
- Example: `A.length`
- NOTE: length is not a method `data.length(`

Auto initialization

- When you create an array in this way: `new int[10];` the elements are initialized t

index	0	1	2	3	4	5	6	7	8	9
A value	0	0	0	0	0	0				

- Each element initially gets a "zero-equivalent

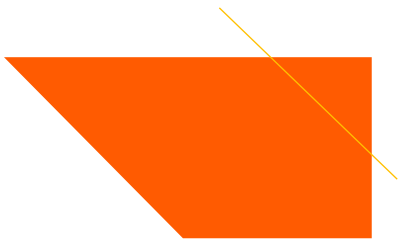
Type	Defa
`int`	`0`
`double`	`0.0`

boolean	false
String	null

Accessing element

- To access the elements in an array, we use th
  `<arrayName>[<index>]`

	index	0	1	2	3	4	5	6	7	8	9
A	value	27	0	0	-6	0	0				



`A[0]` access the first e

`A[3]` access the fourth element

Modifying elemen

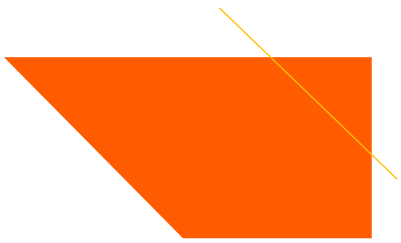
	index	0	1	2	3	4	5	6	7	8	9
A	value	0	0	0	0	0	0				

- To modify an elements in an array, we use th

```
<arrayName>[<index>
```

```
A[0] = 27;

A[3] = -6;
```



A

index	0	1	2	3	4	5	6	7	8	9
value	27	0	0	-6	0	0				

Accessing element

- Legal index values: integers from `0` to `<arra`
- Reading or writing any index outside this rang

  `ArrayIndexOutOfBoundsException`

```
int[] A = new int[10];
System.out.println(A[0]);        //
System.out.println(A[9]);        //
System.out.println(A[-1]);       //
System.out.println(A[10]);       //
```

index	0	1	2	3	4	5	6	7	8	9
A value	0	0	0	0	0					

Accessing element

- The index can be any integer expression: `int`
- We can operate on an array element in the sa other variable of that type
- Example: Applying a 10% late penalty to the c

```
A[i] = (int)(A[i]* 0.9);
```

Another way to cr

- If we know that we want an array to contain sp
  when create the array `int[] data = {7, 8,`
- This list of values is known as an **initialization** l
- We don't use the `new` operator in this case
- We don't specify the length of the array (it is d
  values in the initialization list)

```
double[] heights = {65.2, 72.0, 70.6, 67

boolean[] isPassing = {true, true, false
```

Arrays of other typ

```
double[] results = new double[5];
results[2] = 3.4;
results[4] = -0.5;
```

re

```
boolean[] test = new boolean[6];
test[3] = true;
```

test value

Traversing Arrays

- Often, we will want to do something like wall
  to each cell in the array. We use a for loop:

```
int[] primes = {2, 3, 5, 7, 11, 13, 1

for(int i = 0; i < primes.length; i++
      System.out.println( primes[i])
}
```

Arrays and static m

- **Method declaration**
- Syntax:

```
public static type methodName
```

- Write a
  method that returns the average of the given
  array of numbers

Arrays and static r

- **Method declaration**
- Syntax:

```
public static type methodName
```

- Write a
  method that returns the average of the given

```
public static double average(int[] numbers)
    int sum = 0;
    for (int i = 0; i < numbers.length; i++
       sum += numbers[i];
    }
    return (double) sum / numbers.length;
}
```

- You don’t specify the array’s length (but you c

Arrays and static m

- **Method call**
  - Syntax:

```
methodName(arrayName);
```

- Write a r
  average of the given array of numbers

```
public class MyProgram { public static voic
      main(String[] args) {

          int[] iq = {126, 84, 149, 167, 95
          double avg = average(iq);
          System.out.println("Average IQ =
      }
      ...
```

- Notice that you don't write the `[]` when pass

Arrays and static m

- **Return an array –** method declaration
- Syn

```
public static type[] methodNa
```

- Write a method that returns an array with two
  each value

  `[1, 4, 0, 7] -> [1, 1, 4, 4, 0, 0, 7, 7]`

Arrays and static m

- **Return an array –** method declaration
  - Syn

```
public static type[] methodNa
```

- Write a
  method that returns an array with two copies
  value

  `[1, 4, 0, 7] -> [1, 1, 4, 4, 0, 0, 7, 7]`

```
public static int[] twoCopies(int[] numbers
      int[] result = new int[2 *
      numbers.length]; for (int i = 0; i <
      numbers.length; i++) { result[2 * i]
      numbers[i]; result[2 * i + 1] =
      numbers[i];
      }
      return result;
  }
```

Arrays and static n

- **Return an array –** method call
- Syntax:

```
type[] arrayName =  methodNam
```

- Write a method that returns an array with two
of each value

```
[1, 4, 0, 7] -> [1, 1, 4, 4, 0, 0, 7, 7]
```

```
public class MyProgram { public static void
      main(String[] args) { int[] iq = {126
      84, 149, 167, 95};
          int[] out = twoCopies(iq);
                      System.out.println(Arra
      }
      ...
```

Limitations of arra

- You cannot resize an existing array

```
int[] A = new
int[4]; A.length = 10;
// error
```

- An array does not know how to print itself

```
int[] A1 = {42, -7, 1, 15};
System.out.println(A1);
```

- You cannot compare arrays with `==` or `.equ`

```
int[] A1 = {42, -7, 1, 15}; int[] A2 =
{42, -7, 1, 15}; if (A1 == A2) {  ... }
// false! if (A1.equals(A2)) {  ... }
// false!
```

Limitations of arra

```
public static void main(String[] args)
        int[] A = {126, 167,
        95}; int[] B = A; int[]
        C = {126, 167, 95};

        System.out.println("A location
        System.out.println("B location
        System.out.println("C location

        System.out.println(Arrays.toStr
        System.out.println(Arrays.toStr
        System.out.println(Arrays.toStr
}
```