

Advanced Programming Techniques in Java



COSI 12B



Review: Strings

- "Hello, world!" or "Enter a number:" are *strings*
- Java supplies a class called `String` used to create and process strings
- A *string* is an object storing a sequence of characters
- String objects have
 - Fields (or data values): the characters in the string
 - Methods (or operations): get the length of the string, get a substring, etc.
- Strings in Java are immutable, which means that once they are constructed, their value can never change



Review: Strings

Method name	Description
<code>charAt (index)</code>	Returns the character at the index location in the string
<code>length ()</code>	Returns the number of characters in this string
<code>substring (index1, index2)</code> or <code>substring (index1)</code>	Returns the characters in this string from index1 (inclusive) to index2 (<u>exclusive</u>); if index2 is omitted, grabs till end of string
<code>toLowerCase ()</code>	Returns a new string with all lowercase letters
<code>toUpperCase ()</code>	Returns a new string with all uppercase letters
...	



Review: Value semantic

- **Value semantics** (or value types): behavior where values are copied when assigned, passed as parameters, or returned
- All primitive types in Java use value semantics
- When one variable is assigned to another, its value is copied
- Modifying the value of one variable does not affect others

```
int x = 5;  
  
int y = x;      // x = 5, y = 5  
  
y = 17;         // x = 5, y = 17  
  
x = 8;          // x = 8, y = 17
```



Review: Reference semantic

- If a variable represents an object, the object itself is not stored inside the variable
- The object is located somewhere else in memory, and the variable holds the memory address of the object
 - We say that the variable stores a reference to the object
 - Such variables are called **reference variables (or types)**
- When one variable is assigned to another, the object is not copied; both variables refer to the same object
- Modifying the value of one variable will affect others



Class objectives

- ❖ Random (Section 5.1)
- ❖ Files I/O (Chapter 6)
- ❖ Exceptions (Section 4.4, Chapter 6)



Random Numbers



Random Numbers

- Programs are, by their nature, predictable and non-random but we can produce values that seem to be random
- Pseudorandom numbers** are numbers that are derived from predictable and well-defined algorithms, they mimic the properties of numbers chosen at random



Random Numbers

- Java provides several mechanisms for obtaining pseudorandom number:
- **Call the method `random` from the `Math` class**
 - `Math.random()` gives you a random value of type double between 0.0 (included) and 1.0
 - You can use multiplication to change the range of the numbers the method produces
- **Random class**



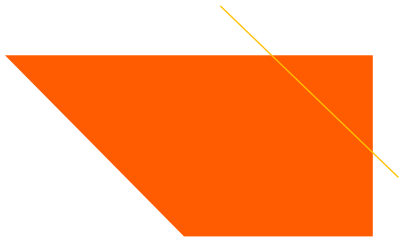
The Random Class

- ❖ A Random object generates pseudorandom numbers
- ❖ The class Random is found in the `java.util` package

```
import java.util.*;
```

Method name	Description
<code>nextInt()</code>	returns a random integer between -2^{31} to $(2^{31} - 1)$
<code>nextInt(max)</code>	returns a random integer in the range $[0, max)$ in other words, 0 to $max-1$ inclusive
<code>nextDouble()</code>	returns a random real number in the range $[0.0, 1.0)$
<code>nextBoolean()</code>	Returns a random logical value of <code>true</code> or <code>false</code>

```
Random rand = new Random();  
int randomNumber = rand.nextInt(10);    // 0-9
```



Random Questions

Given the following declaration: `Random rand = new Random();` how would you get?

1. A random number between 1 and 47 inclusive?
2. A random number between 23 and 30 inclusive?
3. A random even number between 4 and 12 inclusive?
4. A random number between 1.5 and 4.0



Random Questions

Given the following declaration: `Random rand = new Random();` how would you get?

1. A random number between 1 and 47 inclusive?

```
int randomNumber = rand.nextInt(47) + 1;
```

2. A random number between 23 and 30 inclusive?

```
int randomNumber = rand.nextInt(8) + 23;
```

3. A random even number between 4 and 12 inclusive?

```
int randomNumber = rand.nextInt(5) * 2 + 4;
```

4. A random number between 1.5 and 4.0

```
double randomNumber = rand.nextDouble() * 2.5 + 1.5;
```



File Processing & Exceptions



File processing

- To access a file from inside a Java program you need to construct an object that will represent the file
 - Create a **File** object to get info about a file on your drive
 - This doesn't actually create a new file on the hard disk

```
File f = new File("example.txt");  
if (f.exists() && f.length() > 1000) {  
    f.delete();  
}
```



Reading from files

- We can use `Scanner` objects to read from a text file
- We create a `File` object from the file, and pass that object to `Scanner` instead of passing `System.in`

```
File f = new File("example.txt");  
Scanner input = new Scanner(f);
```

```
Scanner input = new Scanner(new File("example.txt"));
```



Compiler error with files

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

public class ReadFile {
    public static void main(String[] args) {
        Scanner input = new Scanner(new File("data.txt"));
        String text = input.next();
        System.out.println(text);
    }
}
```

❖ The program fails to compile with the following error

```
ReadFile.java:6: unreported exception java.io.FileNotFoundException;
must be caught or declared to be thrown
    Scanner input = new Scanner(new File("data.txt"));
                        ^
```




Exceptions

- ✧ An **exception** is an error that occurs at runtime as a result of some type of "exceptional" circumstance
 - ✧ Dividing an integer by 0
 - ✧ Calling substring on a `String` and passing too large an index
 - ✧ Trying to read the wrong type of value from a `Scanner`
 - ✧ Trying to read a file that does not exist

`StringIndexOutOfBoundsException`

`IllegalArgumentException`



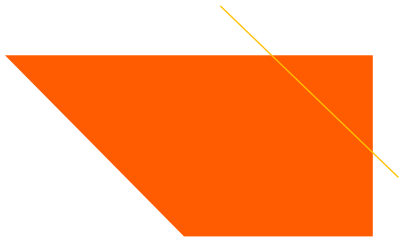
Checked Vs Unchecked Exceptions

❖ *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 defensive programming)
- ❖ a serious external condition that is unrecoverable
- ❖ eg. ArrayIndexOutOfBoundsException



Exceptions

- When using a `Scanner` to process a file, we can get a `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"** an exception
- It is also possible to **"catch"** (handle or fix) an exception
- 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**



The throws clause

- **throws clause** is a keyword on a method's header that state that it may generate an exception (and will not handle it)

Syntax

```
public static <type> <name>(<params>) throws <type> {
```

Example

```
public class ReadFile {  
    public static void main(String[] args) throws FileNotFoundException {
```

- Like saying, "I hereby announce that this method might throw an exception, and I accept the consequences if this happens"