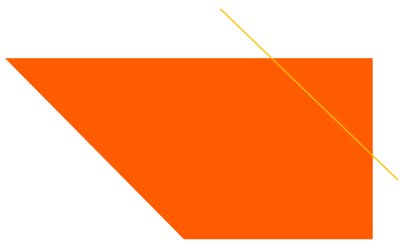


Advanced Programming Techniques in Java



Review: Strings

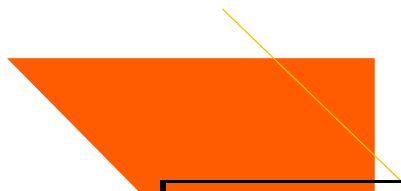
- "Hello, world!" or "Enter a number"
- Java supplies a class called `String` used to create strings
- A *string* is an object storing a sequence of characters
- String objects have
 - Fields (or data values): the characters in the string
 - Methods (or operations): get the length of the string, get the character at a particular position



- Strings in Java are immutable, constructed, their value can never change

Review: Strings

Method name	
<code>charAt(index)</code>	Returns the character at the specified index in the string
<code>length()</code>	Returns the length of the string
<code>substring(index1, index2)</code> or <code>substring(index1)</code>	Returns the substring of the string starting from index1 (inclusive) to index2 (exclusive). If index2 is omitted, the substring extends to the end of the string.



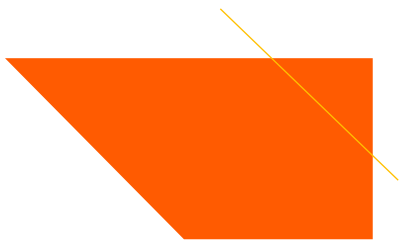
<code>toLowerCase()</code>	Returns a new
<code>toUpperCase()</code>	Returns a new
...	

Review: Value s

- **Value semantics** (or value types): behavior assigned, passed as parameters, or returned use value semantics
- When one variable is assigned to another Modifying the value of one variable does not

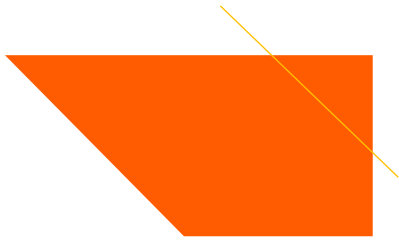


```
int x = 5; int y = x;  
// x = 5, y = 5 y = 17;  
// x = 5, y = 17 x = 8;  
// x = 8, y = 17
```



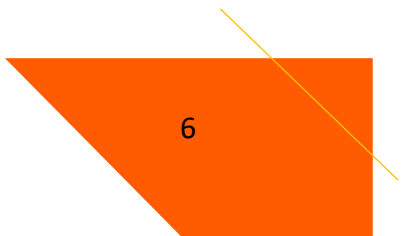
Review: Refer

- If a variable represents an object, the object is located at the memory address stored in the variable
- The object is located somewhere else in memory
 - We say that the variable stores a reference to the object
 - Such variables are called **reference variables**
- When one variable is assigned to another, both variables refer to the same object
- Modifying the value of one variable will affect the other

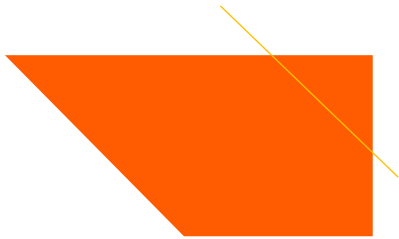


Class objectives

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



Random Nu




Random Number

- Programs are, by their nature, predictable and generate values that seem to be random
- Pseudorandom numbers** are numbers that, using well-defined algorithms, they mimic the properties of random

Random Number

- Java provides several mechanisms for obtaining random numbers
- Call the method `random` from the `Math` class**



`Math.random()` gives you a random number between 0.0 (included) and 1.0

● You can use multiplication to change the range


● 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 0 and <code>Integer.MAX_VALUE</code>



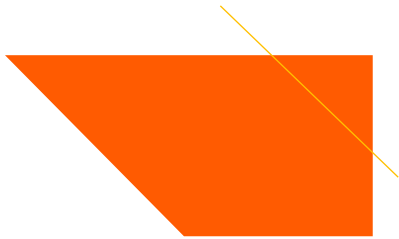
<code>nextInt(max)</code>	returns a random integer in the range 0 to <i>max</i> -1 inclusive. In other words, 0 to <i>max</i> -1 inclusive.
<code>nextDouble()</code>	returns a random real number between 0.0 and 1.0, including 0.0 but not 1.0.
<code>nextBoolean()</code>	Returns a random logical value, either <code>true</code> or <code>false</code> .

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

Random Question

Given the following declaration: `Random r = new Random(47);` what would you get?

1. A random number between 1 and 47 inclusive.



2. A random number between 23 and 47
3. A random even number between 4 and 12
4. A random number between 1.5 and 4.0

Random Question

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

1. A random number between 1 and 47 inclusive
`int randomNumber = rand.nextInt(47);`



2. A random number between 23

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

3. A random even number between 4 and 12

```
int randomNumber = rand.nextInt(12) * 2;
```

4. A random number between 1.5 and 4.0

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



File Processing &



File processing

- To access a file from inside a Java program, you must first construct an object that will represent the file.
- Create a **File** object to get info about a file on the system.
- This doesn't actually create a new file on the system.

```
File f = new File("example.txt");
if (f.exists() && f.length() > 0) {
    f.delete();
}
```



Reading from fi

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

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

```
Scanner input = new Scanner(new File
```




Compiler error v

```
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("input.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("input.txt"));
                                ^
```



Exceptions

- An **exception** is an error that occurs at runtime in an "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 `String`
 - Trying to read a file that does not exist

```
StringIndexOutOfBoundsException  
IllegalArgumentException
```



Check

❖ *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 `try-catch`)

❖ a serious external condition that is unrecoverable

`ArrayIndexOutOfBoundsException`



Exceptions

- When using a `Scanner` to process a file, we can catch 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 indicates the method may generate an exception (and will not handle it)

```
static <type> <name>(<params>) throws  
public class ReadFile { public static void m  
    FileNotFoundException {
```

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