Using Java Classes

String, Math, and Scanner

Strings

- A String is an object that holds a sequence of characters.
- To create a String:
  - **`String team = "Springfield Isotopes";`**
  - **`String sponsor = new String("Duff Beer");`**
- Each character in the string has an index.
- The first character has index 0, the second character has index 1, etc.
- Strings are **immutable**.

Concatenation

- To attach two strings together, we use the process of concatenation, which is represented using the + operator.
- Examples:

```
String team = "Springfield Isotopes";
String sponsor = new String("Duff Beer");
System.out.println("The " + team +
  " are sponsored by " + sponsor);
String headline = team + " Drink " +
  sponsor;
System.out.println(headline);
```

Methods

- Every string has a set of "behaviors" that allow us to perform actions on the string.
- These behaviors are method calls defined in the **Java API**.
- Some methods will require arguments (data) in order to perform their actions.
- See the course website help section for a link to the Java API online.
  - http://java.sun.com/j2se/1.5.0/docs/api/

Length

This is called a method signature.

- $\boxed{\texttt{int length( )}}$
  - **`int`** indicates the data type of the answer that the **`length`** method returns
  - **`length`** is the name of the method
  - **`( )`** indicates that the **`length`** method requires no information to do its job (no arguments)
  - BEHAVIOR: returns the number of characters in this string
- Example:

```
String team = "Springfield Isotopes";
System.out.println(team.length());
```

Substrings

- **`String substring(int startIndex,`**
  `int stopIndex)`
  - this method will return a **`String`** as its result
  - **`substring`** is the name of the method
  - **`(int startIndex, int stopIndex)`** indicates that this method requires two integer arguments to do its job
  - BEHAVIOR: **`substring`** returns a new string consisting of the substring starting at index **`startIndex`** and ending at **`stopIndex`**-1 in this string.

Substrings

This is an example of **overloading** since `substring` is defined 2 different ways.

- **`String substring(int startIndex)`**
  - This method will return a **`String`** as its result
  - **`substring`** is the name of the method
  - **`(int startIndex)`** indicates that this method requires one integer argument to do its job
  - BEHAVIOR: **`substr`** returns a new string consisting of the substring starting at index **`startIndex`** and ending with the last character in this string.

7

Substrings

- Examples:

```
String team = "Springfield Isotopes";
String sponsor = new String("Duff Beer");
// print out:  Go Isotopes!
System.out.println("Go " +
  team.substring(12) + "!");
// print out:  You can't get enough
//             of that wonderful Duff!
System.out.println("You can't get enough\n"
    + "of that wonderful "
    + sponsor.substring(0,4) + "!");
```

8

Getting a single character

- **`char charAt(int index)`**
  - this method will return a **`char`** as its result
  - **`charAt`** is the name of the method
  - **`(int index)`** indicates that this method requires one integer argument to do its job
  - BEHAVIOR: **`charAt`** returns the character located at the given **`index`**
- Example:

```
String team = "Springfield Isotopes";
char teamLetter = team.charAt(12);
```

9

Replacing characters

- **`String replace(char oldChar, char newChar)`**
  - this method will return a **`String`** as its result
  - **`replace`** is the name of the method
  - **`(char oldChar, char newChar)`** indicates that this method requires two character arguments
  - BEHAVIOR: **`replace`** returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.

```
String team = "Springfield Isotopes";
System.out.println(team.replace('i','y'));
```

10

Converting case

- **`String toUpperCase()`**
  - **returns a new string with all letters of this string converted to uppercase**
- **`String toLowerCase()`**
  - **returns a new string with all letters of this string converted to lowercase**
- Examples:

```
String sponsor = new String("Duff Beer");
System.out.println(sponsor.toUpperCase());
System.out.println(sponsor.toLowerCase());
```

11

Sequencing methods

```
String sponsor = new String("Duff Beer");
```

What is the output of each of these?

```
System.out.println(
   sponsor.toUpperCase().substring(5));
System.out.println(
   sponsor.substring(5).toUpperCase());
System.out.println(
   sponsor.substring(sponsor.length()-1));
System.out.println(
   sponsor.length().toLowerCase());
```

12

The Math class

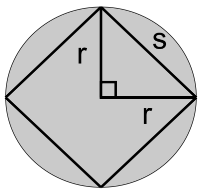
- The `Math` class contains methods that perform common mathematical operations.
- Signatures:

```
static double ceil(double num)
static double floor(double num)
static double sqrt(double num)
static double pow(double num, double power)
```

The `static` keyword in the signature indicates that we call this method using the name of the class itself (`Math`).

13

Examples



```
double area = Math.PI * radius * radius;
double circumference = 2.0 * Math.PI * radius;
double s = Math.sqrt(2.0 * Math.pow(r, 2.0));
double squareAreaLB = Math.floor(s * s);
double squareAreaUB = Math.ceil(s * s);
```

14

Generating Random Numbers

- The `Math` class has a random method that generates a random double in [0,1).
  - The number isn't really truly random. It's pseudo-random.
  - The number is uniformly-distributed in the range.
- To generate a random number , we might write:

```
double randNum = Math.random();
```

15

Generating Random Numbers

- If we want to generate a random number in another range, we can scale (multiply) and/or translate (add) to the random number to get the desired range.
- Generate a random double in [0,15):

```
double randNum = Math.random() * 15.0;
```

- Generate a random double in [15,100):

```
double randNum = Math.random()
                       * 85.0 + 15.0;
```

16

Generating Random Integers

- If we want to generate a random integer, we can generate a number using `Math.random` in the proper range using multiplication and addition, and then use typecasting to get an integer.
- Generate a random integer in {0, 1, ..., 14}

```
int randNum =
(int)(Math.random() * 15.0);
```

generates a random number between 0.0 and 14.99999...

truncates the random number to an `int` between 0 and 14.

17

Generating Random Integers

- Example: Generate a random multiple of 5 between 5 and 100 (inclusive)

```
Math.random()                       [0,1)
Math.random() * 20.0                [0,20)
(int)(Math.random() * 20.0)         0, 1, ..., 19
(int)(Math.random() * 20.0) + 1
                                    1, 2, ..., 20
((int)(Math.random() * 20.0) + 1) * 5
                                    5, 10, ..., 100
```

18

Generating Random Integers

- What's wrong with the following statement that generates a random multiple of 5 in the range between 5 and 100 (inclusive)?

```
int randNum =
   ((int)Math.random() * 20.0 + 1) * 5
```

19

Program Input
using Java 5

- Java 5 provides a class called **`Scanner`** to allow us to read data from the keyboard into our program while it's running.
  - Scanner is in the **`java.util`** package (classes are organized in packages).
  - Since the compiler normally does not check the **`java.util`** package during compilation, we need to **`import`** this package so the compiler can find Scanner and any methods we use in order to see if we're using them correctly (based on syntax).

```
import java.util.*;
public class MyProgram { …
```

20

Creating a Scanner

- Before we write instructions to read data from the keyboard when our program runs, we must create a Scanner object first:

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

`scan` is the name of the **`Scanner`** that we have created

`System.in` represents our input device (i.e. the keyboard)

21

Some Scanner Methods

- **`String nextLine()`**
  BEHAVIOR: read in and return a **`String`** containing all text up to the next "return" key
- **`int nextInt()`**
  BEHAVIOR: read in and return an **`int`** input from the user
- **`double nextDouble()`**
  BEHAVIOR: read in and return a **`double`** input from the user

22

Scanner Example

```
Scanner scan = new Scanner(System.in);
System.out.println(
  "Please input your birth month:");
String month = scan.nextLine();
System.out.println(
  "Please input your birth day number:");
int dayNumber = scan.nextInt();
```

What happens if we try to read in an integer but the user doesn't give us an integer?

23