Strings & things

Introduction to the Java API

The Java API

• API = Applications Programming Interface
  – Rich set of predefined classes and objects for use in Java programs
  – Many aspects of API require import statements for access
  – Classes and objects defined in the package java.lang are accessible without the import statement; two such classes are String and Math

The String class

• The String class describes operations that can be performed on, and by, String objects
• A String object is a variable of type String, initialized with a String literal
• Example:
  
  String name = new String ("Cate Sheller");
  String variable String literal
Object variables & constructors

- The example on the previous slide introduces a new syntactic pattern
  - Strings are objects
  - Objects must be instantiated
  - This is accomplished via the new operator and a call to a constructor, a special kind of method
    - The constructor has the same name as the class
    - It is used to create a new instance of the class – i.e. an object

General Syntax for Objects

ClassType objectName = new ClassType(arg(s));
- “ClassType” is the name of a class – either from the Java API or programmer-defined
- “objectName” is the name you have chosen for your variable
- “arg(s)” – 0 or more arguments may be required; for example, when a String object is instantiated, the required argument is a String literal

The String class is exceptional

- Although everything on the two previous slides is true, it is worth noting that Strings can behave differently from most objects
- When a String variable is declared, you can instantiate an object with only an implied call to the constructor, as in the example below:
  String aWord = “word”;
- Most objects don’t behave this way; for consistency, it is best to learn the method described previously
String operations: concatenation

- We have already seen that the + operator can be used to concatenate String literals; this operator can be used on String variables as well, as in this example:

```java
String name = new String("Cute Sheller");
String myFave = new String("Favorite professor");
String myFaveName = new String(myFave + ";" + name);
```

String operations: assignment

- A String variable can be assigned:
  - The value of a String literal
  - A String expression (e.g. a concatenated String)
  - Another String variable
- There are some important differences between the first two operations and the last one, but we’ll talk about that later

String methods

- Like most classes, the String class contains several member methods that can be called from String objects (variables)
- Several of these are listed and described on pages 38 – 42 of your textbook; we will examine some of these
String methods: substring

• **substring**: takes 2 arguments representing the beginning and ending positions of a String within a String – returns the resulting substring
  
  – Note that the first position in a String in Java is designated position 0 – so a 4-letter word would start at 0 and end at 3
  
  – An error will result if you attempt to call the method using positions that don’t exist within the String object

Examples using substring

```java
String bigRiver = new String(“Mississippi”);
bigRiver.substring(6, 9) // returns “sip”
bigRiver.substring(0, 3) // returns “Mis”
bigRiver.substring(4, 6) // returns “is”

• Note that the **first argument indicates the starting position** of the substring, while the **second argument indicates the position after the end of the substring**
```

Examples using substring

• Method calls like those in the example would return the literal values indicated, and would usually occur within the context of an assignment statement or another method call;

```
String sub = new String(bigRiver.substring(6, 9));
// returns “sip” and assigns it to new object sub
System.out.println(bigRiver.substring(4, 6));
// displays “is” on the output window
```
String methods: length

- The `length` method returns the length (in characters) of the String object; for example, if String `bigRiver` contains the value “Mississippi” then
  ```java
  bigRiver.length() // returns 11
  ```

String methods: `indexOf`

- The `indexOf` method returns a number indicating the position of the beginning of the first occurrence of the substring specified in the message’s argument; examples:
  ```java
  bigRiver.indexOf("Miss") // returns 0
  bigRiver.indexOf("is") // returns 1
  bigRiver.indexOf("sis") // returns 3
  ```

Program example

```java
public class StrNotes {
  public static void main(String [] args) {
    final String NAME = new String ("Cafe");
    String frag = new String (NAME.substring(1, NAME.length()));
    String nonsns1 = new String ("Bo-o-");
    String nonsns2 = new String ("Fana f-a-f-");
    String nonsns3 = new String ("Fe fi mo-m");
    char space = ";

    System.out.println (NAME + space + NAME + space + nonsns1 + frag);
    System.out.println (nonsns2 + frag);
    System.out.println (nonsns3 + frag + space + NAME);
  }
}
```
String Methods: charAt

- charAt
  - Takes int argument representing a position within the calling String object
  - Returns the char value found at that position
  - Valid positions are 0 through length – 1
  - Example:
    ```java
    String name = "Cate";
    char firstLetter = name.charAt(0);
    // firstLetter now contains 'C'
    ```

String methods: changing case

- The methods toUpperCase and toLowerCase each return a String that is the ALL CAPS or all lowercase version of the calling String object
- Neither method changes the calling object
- Example:
  ```java
  String sample = "This is a test";
  System.out.println(sample.toUpperCase());
  // prints THIS IS A TEST – leaves sample unchanged
  ```

The Math Class

- Another standard class from the Java API is the Math class
- Unlike the String class, most of the methods of Math are class methods, not instance methods
- This means that:
  - You don’t need to create a Math object to call them
  - They are called from the Math class itself, rather than from an object
Calculations using Java’s Math class

• The standard Java class Math contains class methods and constants that are useful in performing calculations that go beyond simple arithmetic operations
• The constants defined in the Math class are Math.PI and Math.E, which are defined values for π and e (the base for natural logs), respectively

Math class methods

• Math.abs(a): returns the absolute value of its argument (a), which can be of type int, long, float, or double
• Math.sin(a): returns the sine of its argument, a double value representing an angle in radians; similar trigonometric functions include Math.cos(a) for cosine, Math.tan(a) for tangent, Math.acos(a), Math.asin(a) and Math.atan(a), which provide arccosine, arcsine, and arctangent, respectively

Math class methods

• Math.toDegrees(a): converts a, a double value representing an angle in radians, to the corresponding value in degrees
• Math.toRadians(a): converts a, a double value representing an angle in degrees to the corresponding value in radians
Math class methods

- `Math.sqrt(a)`: returns the square root of a, a value of type double
- `Math.cbrt(a)`: returns the cube root of a, a value of type double
- `Math.pow(a, b)`: returns the value of $a^b$
- `Math.log(a)`: returns the natural log of a, a double value
- `Math.log10(a)`: returns the log base 10 of a, a double value

Math class methods

- `Math.round(a)` takes either a double or float argument, and returns the closest long (if the argument was double) or int (for a float argument) to the value of the argument
- Note that this is different from a type cast – the value returned is a whole number, but it may be rounded up instead of down (as casting always does)
- These and several other Math class methods are described in your text on pages 263-265

Example

```java
// computing the roots of a quadratic equation:
double a, // coefficient of x squared
      b, // coefficient of x
c, // 3rd term in equation
      x1, // first root
      x2; // second root

// read in values for a, b, and c – not shown here ...

x1 = (-b + Math.sqrt(Math.pow(b, 2) – (4 * a * c))) / (2 * a);
x2 = (-b - Math.sqrt(Math.pow(b, 2) – (4 * a * c))) / (2 * a);
```
More Java API standard classes

- Classes Math and String are part of a standard library of classes that are available by default to all Java programs
- Many other classes, such as the Random class, can also be made available, but an additional step is required
- Access to the library containing Random is attained via an `import statement`

Importing Java packages

- A package is a collection of classes; many such packages are available for your use in the Java API
- An import statement gives access to a package
  - The statement below gives access specifically to the Random class:
    ```java
    import java.util.Random;
    ```
  - The statement below provides access to all classes in the java.util package:
    ```java
    import java.util.*;
    ```
- Import statements appear at the top of a program file, before the class heading

Generating random numbers

- Random numbers are useful in programs to simulate occurrence of chance events
- For example, we might use a random number generator to help us simulate the roll of dice or the dealing of a card
- The java.util package contains the Random class, which provides a blueprint for a random number generating object
Generating random numbers

- To create a random number generator, use code like the example below:
  
  Random rg = new Random();

- Once the object is created, you can use it to generate random double or int values, as shown below:
  
  int randomInt = rg.nextInt();
  double randomDbl = rg.nextDouble();

---

Generating random numbers: example program

```java
import java.util.*;

public class Numbers {
    public static void main (String [] args) {
        int rint;
        double rdbl;
        Random randGen;
        randGen = new Random();
        rdbl = Math.abs(randGen.nextDouble());
        System.out.println("Here is a random real number: "+rdbl);
        rint = Math.abs(randGen.nextInt());
        System.out.println("rint is "+rint);
        rint = rint % 10 + 1;
        System.out.println("Here is an integer between 1 and 10: "+rint);
    }
}
```

---

Notes on random numbers

- As the previous slide illustrates, some manipulation is required to ensure that the number generated lies within a particular range
  
  - By default, the nextDouble method returns a value between 0.0 and 1.0
  
  - By default, the nextInt method simply returns a whole number – it may be positive or negative, and could have any value within the int range
Notes on random numbers

- An alternative version of the nextInt method makes the chore of obtaining a positive number within a particular range
- This version of nextInt takes an int argument, that specifies a value that any number generated must be less than

Examples

- If rg is a previously-constructed Random object, then the following expressions produce the values indicated:
  - `rg.nextInt(10)` produces a value between 0 and 9
  - `rg.nextInt(10) + 1` produces a value between 1 and 10
  - `2 * (rg.nextInt(10) + 1)` produces an even number between 2 and 20
  - `rg.nextInt(21) – 10` produces a number between -10 and 10