CPSC 324
Topics in Java Programming

Lecture 4

Agenda

• More on basic language constructs
  – Strings (cont.)
  – Using Classes and Objects
    • We’ll cover the details of classes and objects in later lectures
  – Reading User Input, File I/O
  – Exception Handling

• Reading Assignments
  – Core: Rest of Ch. 3: 58-71

• Homework Assignment #1
  – Due Thurs.
Java Strings

• A String is a “blend” of object and primitive value
  – Strings are really objects
  – But they have a special representation (using “”)
  – Strings in Java denote sequences of Unicode characters

• Examples

  String empty = “”; // an empty string
  String shortstr = “a”; // a string of one char
  String course = “CPSC 324”; // a string of multiple chars
  String title = “Topics in Java Programming”;
Testing String Equality

- Like arrays, an object variable is a reference to the object
- Be careful when checking if two String objects are equal
- Always use the String “dot equals” method

```java
String course = "CPSC 324";
String dept = "CPSC";
if(course == "CPSC 324")
    // probably true bec. of sharing
if(course.substring(0, 4) == dept)
    // probably false (not shared)
if(dept.equals(course.substring(0, 4)))
    // always true
```

Java Strings

- Strings are immutable
  - Once a String object is created, it cannot be modified!
  - For efficiency reasons
    - common strings can be shared
    - can reduce memory use (same string only stored once in memory)
  - Also helps with concurrency (threads)
- This means that operations on String objects create new String objects
  - This was the case for concatenation (+)
  - str1 + str2 creates a new String object
String Builder

- As you build strings using concatenation (e.g., “+”)
  - You may end up with many intermediate objects
  - This can be inefficient (time and space)
  - You can use the StringBuilder class instead …

- The StringBuilder class represents a mutable string

```java
StringBuilder aBuilder = new StringBuilder(); // create obj
aBuilder.append("CPSC"); // add string
aBuilder.append(‘ ‘); // add char
aBuilder.append(3); // add int
aBuilder.append("24"); // add string
String title = aBuilder.toString(); // get string
```

Using Objects and Classes

We have used classes
- Our hello world program
- Static fields (constants)
- Static methods

And now we have started using objects
- String objects
- StringBuilder objects

- We’re going to talk a bit more about using objects
  - Mainly so you can use them in your assignment
- Later we will go into all the gory details!
### Classes versus Objects

A Class is like a “blueprint” for creating objects

- The non-static fields state what each object will have
  - Sets the object “structure”
  - These fields are also called “instance variables” or “state”
  - The static fields are sometimes called “class variables”

- The non-static methods are called on each object

All cards in a card catalog might look like this ...

<table>
<thead>
<tr>
<th>Name:</th>
<th>Phone:</th>
<th>eMail:</th>
</tr>
</thead>
<tbody>
<tr>
<td>___________</td>
<td>_______</td>
<td>___________</td>
</tr>
</tbody>
</table>

The Card Class might be defined like this ...

```java
public class Card {
    public String name;
    public String phone;
    public String eMail;

    public void sendEmail(String msg) {
        ...
    }
}
```

### Using Classes and Objects

- An object is an “instance” of a Class
  - Objects contain values for each non-static field
    - That is, values for each instance variable
  - Objects can have non-static methods called on them
    - These can access instance variables and other instance methods
    - These methods are sometimes called “behaviors”

All cards in a card catalog might look like this ...

<table>
<thead>
<tr>
<th>Name:</th>
<th>Phone:</th>
<th>eMail:</th>
</tr>
</thead>
<tbody>
<tr>
<td>___________</td>
<td>_______</td>
<td>___________</td>
</tr>
</tbody>
</table>

Specific cards in a card catalog might look like this ...

<table>
<thead>
<tr>
<th>Name:</th>
<th>Phone:</th>
<th>eMail:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polly Morphism</td>
<td>555-0343</td>
<td><a href="mailto:pm@reallysmart.com">pm@reallysmart.com</a></td>
</tr>
</tbody>
</table>

We can ask the card to sendEmail
Using Classes and Objects

- Use “new” to create objects from classes
  
  ```java
  Card aCard = new Card(); // parens required here!
  ```

- Use the “dot operator” to set fields and call methods
  ```java
  aCard.name = “Polly Morphism”;
  aCard.phone = “555-0434”;
  aCard.eMail = “pm@reallysmart.com”;
  aCard.sendEmail(“Hi!”);
  ```

- Object variables reference objects (more later…)

```
Using Classes and Objects – Encapsulation

- We usually make instance variables private
  – not visible or accessible outside the object/class

- And add “getter” and “setter” instance methods
  ```java
  public class Card {
      private String name;
      ...
      public void setName(String aName) {
          name = aName;
      }
      public String getName() {
          return name;
      }
  }
  ```
```

Call aCard.setName(“Polly”) to set the value of the name variable
Call aCard.getName() to get the value of the name variable
Calling aCard.name (outside of the Card class) will give a compile error!
Using Classes and Objects

• Special “constructor” methods initialize objects
• These methods are called “constructors”

```java
public class Card {
    public Card() {
        ...
    }
}
```

• This is called when a new object is created

```java
Card aCard = new Card(); // calls the Card() constructor
```

• The compiler always provides a “default” constructor if no constructors are given
  – The provided constructor takes no arguments

Using Classes and Objects

• Constructors can also take arguments

```java
public class Card {
    public Card() {...}
    public class Card(String aName, String aPhone, String anEmail) {
        name = aName;
        phone = aPhone
        eMail = anEmail;
    }
}
```

```java
Card aCard = new Card(“polly”, “555-0343”, “pm@reallysmart.com”);
```

• We’ll talk more about how to define classes later
Exercise 3

What is the result of this?

```java
public class DVDPlayer {
    private boolean recordable = false;
    public boolean isRecordable() {
        return recordable;
    }

    public void setRecordable(boolean canRecord) {
        recordable = canRecord;
    }

    public void recordDVD() {
        System.out.println("Recording DVD");
    }
    public void playDVD() {
        System.out.println("Playing DVD");
    }

    public static void main(String[] args) {
        DVDPlayer d = new DVDPlayer();
        d.setRecordable(true); // OLD: d.recordable = true
        d.playDVD();
        if(d.isRecordable() == true) // OLD: if(DVDPlayer.isRecordable() == true)
        d.recordDVD();
    }
}
```
Reading from Standard Input

• There are many ways to do I/O in Java …

    Scanner in = new Scanner(System.in);
    String line = in.nextLine();

• For this to work, you’ll need to add an “import” to the top of your class file

    import java.util.Scanner;
    – or (less preferred) –
    import java.util.*;  // imports all classes in util ...

Scanner example

import java.util.Scanner;
public class ScannerTest {
    public static void main(String[] args) {
        System.out.println("What is your name?");
        Scanner in = new Scanner();
        String name = in.nextLine();
        System.out.println("Hi " + name + ", Hi!");
    }
}
Basic File I/O

- We can still use scanner …

```java
File aFile = new File("file.txt"); // create a file object
Scanner in = new Scanner(aFile);  // scanner for file
while(in.hasNextLine()) {
    String line = in.nextLine();   // get a line
    System.out.println(line);       // print the line
}
in.close();                       // close the scanner
```

- See the Scanner API for more operations

Scanner example

```java
import java.util.Scanner;
import java.io.File;
public class ScannerTest {
    public static void main(String[] args) {
        File aFile = new File("file.txt");
        Scanner in = new Scanner(aFile);
        while(in.hasNextLine())
            System.out.println(in.nextLine());
    }
in.close();
}
```

We have a problem!
Throwing Exceptions

• Methods (even constructors) can “throw exceptions”

```java
public Scanner(File source) throws FileNotFoundException {
    ... do stuff
    if (source is not a valid file)
        throw new FileNotFoundException(...); // exception object
    ... everything ok, keep doing stuff
}
```

• Throwing an exception is a special form of control flow
  – Causes method to terminate
  – Passing the exception on to the calling method
  – The calling method can handle the exception, or pass it on

Scanner example

```java
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
public class ScannerTest {
    public static void main(String[] args) throws FileNotFoundException {
        File aFile = new File("file.txt");
        Scanner in = new Scanner(aFile);
        while (in.hasNextLine())
            System.out.println(in.nextLine());
        in.close();
    }
}
```

A quick solution

Here we are "passing on" the exception
**Scanner example**

import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

public class ScannerTest {
    public static void main(String[] args) throws FileNotFoundException {
        File aFile = new File("file.txt");
        Scanner in = new Scanner(aFile);
        while(in.hasNextLine())
            System.out.println(in.nextLine());
        in.close();
    }
}

But, what happens if file.txt is not found?

```
java.io.FileNotFoundException: file.txt (No such file or directory)
at java.io.FileInputStream.open(Native Method)
at java.io.FileInputStream.<init>(FileInputStream.java:106)
at java.util.Scanner.<init>(Scanner.java:621)
at ScannerTest.main(ScannerTest.java:8)
```
Try-catch-blocks

• The basic syntax:

```java
try {
    // statements that may throw exceptions
} catch(ExceptBonType e) {
    // an exception was thrown
    // try to handle the exception
}
// if we don’t exit from the catch statement, we keep going
```

• Things can get more complicated than this

Handling Exceptions in Try-catch-blocks

• Sometimes, print an error and exit program
• Often in practice, handle by ignore and continue
• … but it is good practice to print the stack trace:

```java
try {
    // statements that may throw exceptions
} catch(ExceptBonType e) {
    e.printStackTrace(); // method defined on exceptions
}
// keep going as if nothing happened (if possible)
```

• In the previous (unhandled) exception, the JVM printed the stack trace for us