ArrayList, Wrapper Classes, and Text Processing

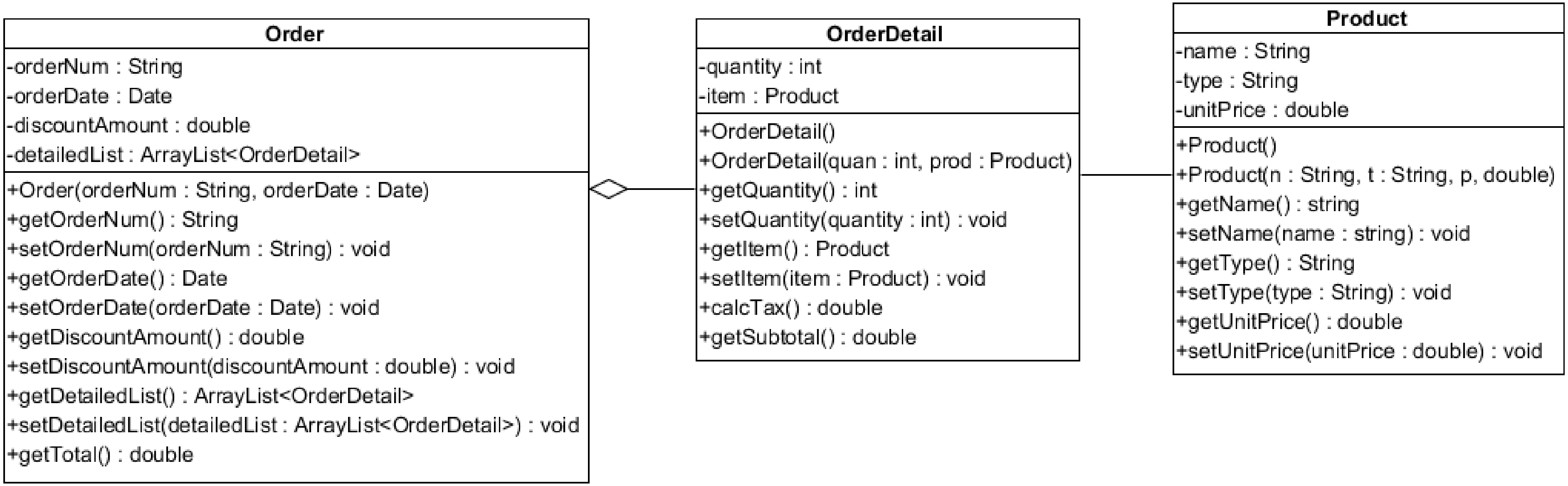
Tony Gaddis (2019) Starting Out with Java: From Control Structures through Data Structures, 4th Edition

Outline

- Class Aggregation
- More on ArrayList Class
- Introduction to Wrapper Classes
- Dialog Boxes
- Converting Strings and Numbers
- Converting a Number to a String
- Text Processing the **`Character`** and **`String`** Methods

UNLV

Aggregation of Objects



Order

-orderNum : String
-orderDate : Date
-discountAmount : double
-detailedList : ArrayList<OrderDetail>

+Order(orderNum : String, orderDate : Date)
+getOrderNum() : String
+setOrderNum(orderNum : String) : void
+getOrderDate() : Date
+setOrderDate(orderDate : Date) : void
+getDiscountAmount() : double
+setDiscountAmount(discountAmount : double) : void
+getDetailedList() : ArrayList<OrderDetail>
+setDetailedList(detailedList : ArrayList<OrderDetail>) : void
+getTotal() : double

OrderDetail

-quantity : int
-item : Product

+OrderDetail()
+OrderDetail(quan : int, prod : Product)
+getQuantity() : int
+setQuantity(quantity : int) : void
+getItem() : Product
+setItem(item : Product) : void
+calcTax() : double
+getSubtotal() : double

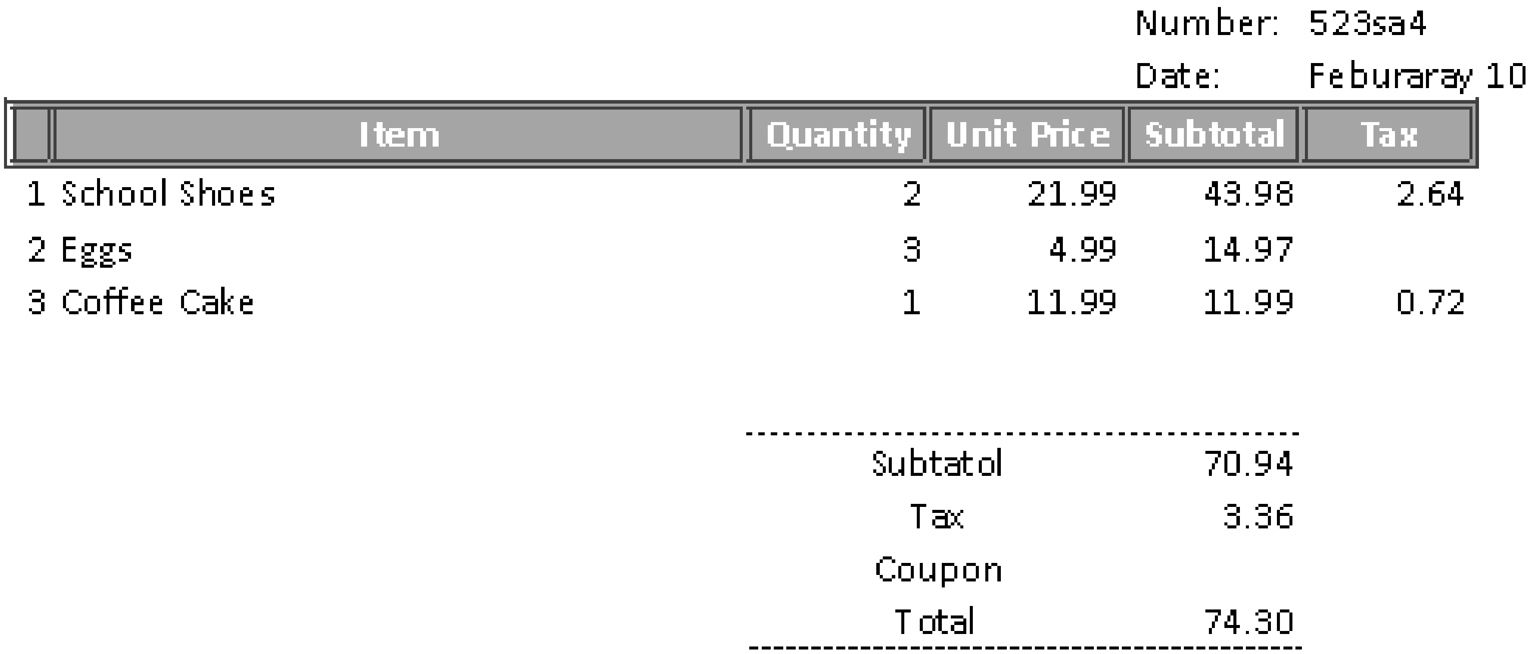
Product

-name : String
-type : String
-unitPrice : double

+Product()
+Product(n : String, t : String, p, double)
+getName() : string
+setName(name : string) : void
+getType() : String
+setType(type : String) : void
+getUnitPrice() : double
+setUnitPrice(unitPrice : double) : void

- The field of a Class can be an array or an **`ArrayList`**
  - An aggregation has a diamond end pointing to the part containing the whole.
- The aggregation of order details is an order.
- Total is determined by the subtotal and the discount/coupon amount entered.

Example: Sales Receipt

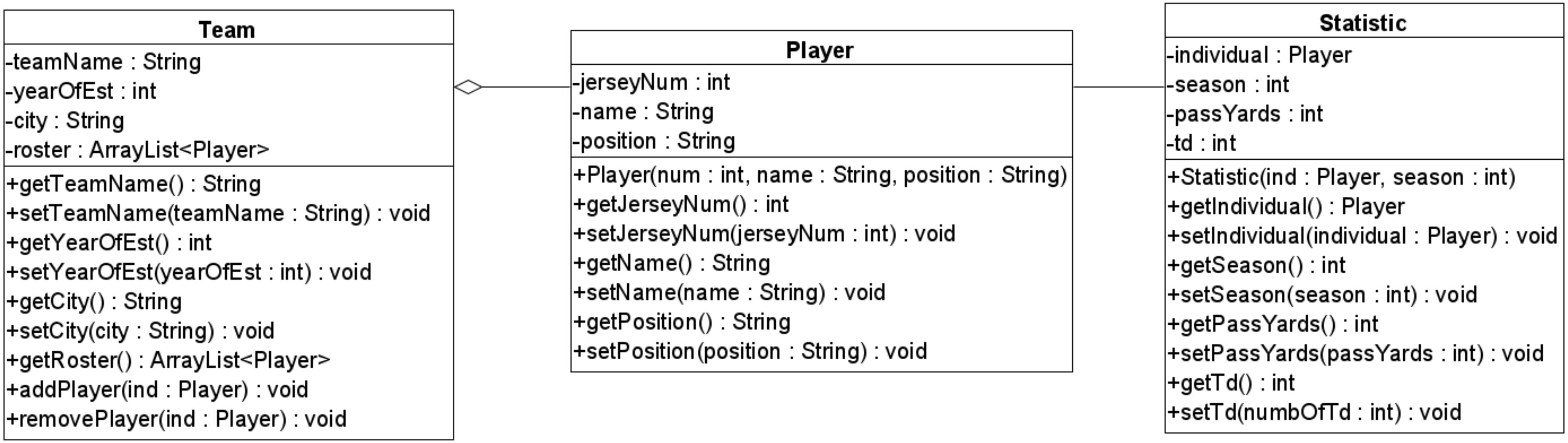


Number:	523sa4
Date:	Feburaray 10

	Item	Quantity	Unit Price	Subtotal	Tax
1	School Shoes	2	21.99	43.98	2.64
2	Eggs	3	4.99	14.97	
3	Coffee Cake	1	11.99	11.99	0.72

Subtatol	70.94
Tax	3.36
Coupon	
Total	74.30

Aggregation Example



Team

-teamName : String
-yearOfEst : int
-city : String
-roster : ArrayList<Player>

+getTeamName() : String
+setTeamName(teamName : String) : void
+getYearOfEst() : int
+setYearOfEst(yearOfEst : int) : void
+getCity() : String
+setCity(city : String) : void
+getRoster() : ArrayList<Player>
+addPlayer(ind : Player) : void
+removePlayer(ind : Player) : void

Player

-jerseyNum : int
-name : String
-position : String

+Player(num : int, name : String, position : String)
+getJerseyNum() : int
+setJerseyNum(jerseyNum : int) : void
+getName() : String
+setName(name : String) : void
+getPosition() : String
+setPosition(position : String) : void

Statistic

-individual : Player
-season : int
-passYards : int
-td : int

+Statistic(ind : Player, season : int)
+getIndividual() : Player
+setIndividual(individual : Player) : void
+getSeason() : int
+setSeason(season : int) : void
+getPassYards() : int
+setPassYards(passYards : int) : void
+getTd() : int
+setTd(numbOfTd : int) : void

- A team object will have a list of players
- A statistic object is about one player in a certain season
  - One Player object may be associated with many Statistic objects (one for each season).

Lab (1)

- Sales Receipt
  - Order.java
  - OrderList.java
  - Product.java

More about ArrayList Class

- ArrayList Class works on objects only
  - Cannot store values of primitive data types(such as int, double, char, and long)
  - ArrayList can hold String objects
  - ArrayList can hold wrapper class objects (Double, Integer).

Wrapper Classes (1)

- Java provides 8 primitive data types.
  - **`byte`**
  - **`short`**
  - **`int`**
  - **`long`**
  - **`float`**
  - **`double`**
  - **`boolean`**
  - **`char`**
  - They are called “primitive” because they are not created from classes.
- Java provides wrapper classes for all of the primitive data types.
  - A *wrapper class* is a class that is “wrapped around” a primitive data type.
  - The wrapper classes are part of **`java.lang`** so to use them, there is no **`import`** statement required.

Wrapper Classes (2)

- Java provides wrapper classes for all of the primitive data types.
- The numeric primitive wrapper classes are:

Wrapper Class	Numeric Primitive Type It Applies To
Byte	byte
Double	double
Float	float
Integer	int
Long	long
Short	short

Wrapper Classes (3)

- Wrapper classes allow you to create objects to represent a primitive.
- Wrapper classes are immutable
  - Once you create an object, you cannot change the object’s value.
- Wrapper classes provide static methods that are very useful

Autoboxing and Unboxing (1)

- You can declare a wrapper class variable and assign a value:

```
Integer number;
number = 7;
```

- You may think this is an error, but because number is a wrapper class variable, *autoboxing* occurs.
- *Unboxing* does the opposite with wrapper class variables:

```
Integer myInt = 5;          // Autoboxes the value 5
int primitiveNumber;
primitiveNumber = myInt;  // unboxing
```

Autoboxing and Unboxing (2)

- You rarely need to declare numeric wrapper class objects, but they can be useful when you need to work with primitives in a context where primitives are not permitted
- Recall the **`ArrayList`** class, which works only with objects.

```
ArrayList<int> list =
      new ArrayList<int>();      // Error!
ArrayList<Integer> list =
      new ArrayList<Integer>(); // OK!
```

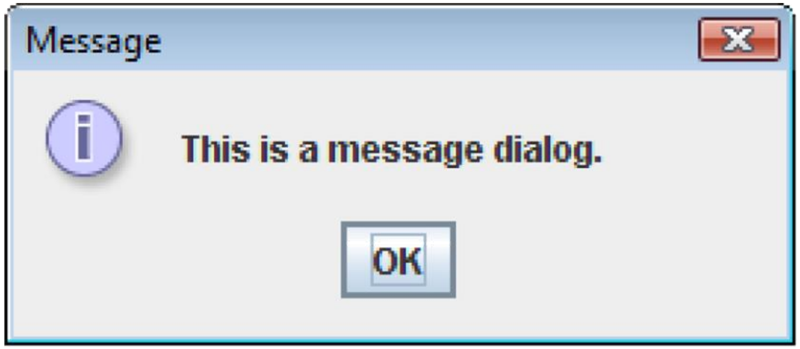
- Autoboxing and unboxing allow you to conveniently use **`ArrayLists`** with primitives.

Dialog Boxes

- A *dialog box* is a small graphical window that displays a message to the user or requests input.
- A variety of dialog boxes can be displayed using the **`JOptionPane`** class.
- Two of the dialog boxes are:
  - Message Dialog - a dialog box that displays a message.
  - Input Dialog - a dialog box that prompts the user for input.

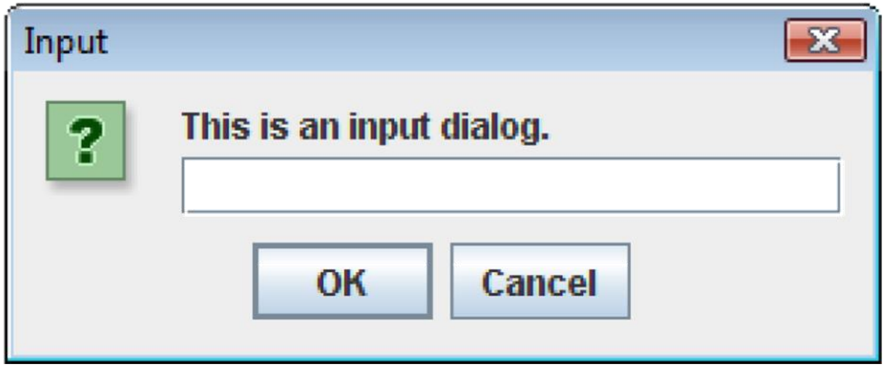
JOptionPane Class (1)

Message dialog



This is a message dialog.

Input dialog



This is an input dialog.

`JOptionPane` Class (2)

- The **`JOptionPane`** class is not automatically available to your Java programs.
- The following statement must be before the program’s class header:

```
import javax.swing.JOptionPane;
```

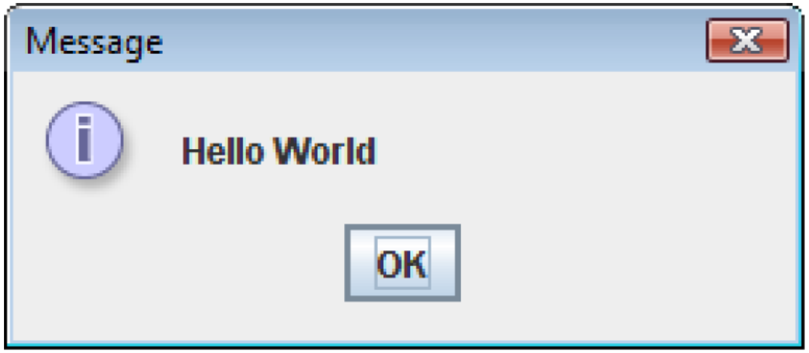
- The **`JOptionPane`** class provides methods to display each type of dialog box.

Message Dialogs

- **`JOptionPane.showMessageDialog`** method is used to display a message dialog.

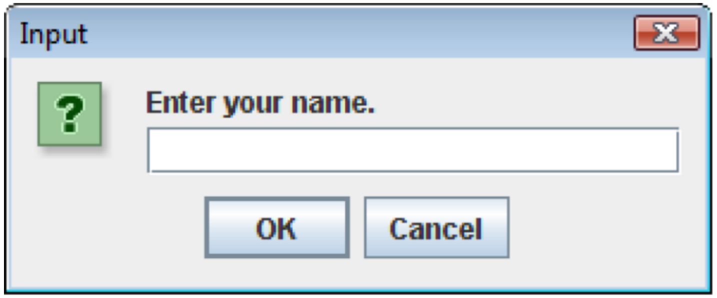
```
JOptionPane.showMessageDialog(null, "Hello
  World");
```

  - The first argument will be discussed in later in this semester.
  - The second argument is the message that is to be displayed.



Input Dialogs (1)

- An input dialog is a quick and simple way to ask the user to enter data.
  - The dialog displays a text field, an Ok button and a Cancel button.
  - If Ok is pressed, the dialog returns the user’s input.
  - If Cancel is pressed, the dialog returns null.

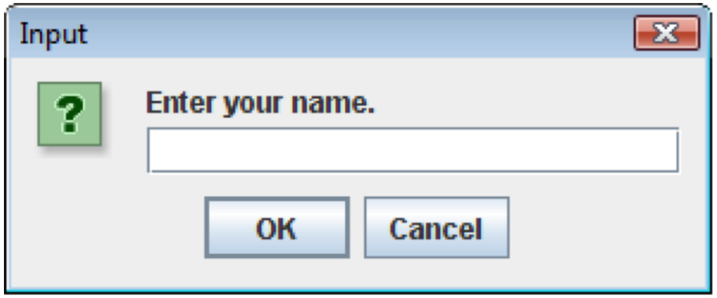


Enter your name.

Input Dialogs (2)

```
String name;
name = JOptionPane.showInputDialog(
         "Enter your name.");
```

- The argument passed to the method is the message to display.
- If the user clicks on the OK button, **`name`** references the string entered by the user.
- If the user clicks on the Cancel button, **`name`** references **`null`**.



Enter your name.

`System.exit()` Method

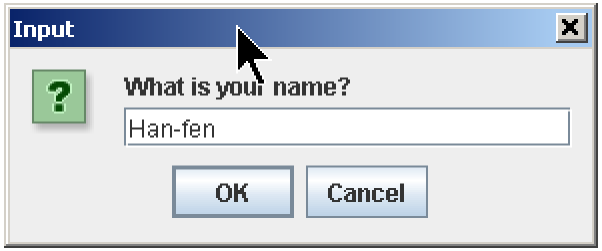
- A program that uses **`JOptionPane`** does not automatically stop executing when the end of the main method is reached.
- Java generates a *thread*, which is a process running in the computer, when a **`JOptionPane`** is created.
  - If the **`System.exit`** method is not called, this thread continues to execute.
- The **`System.exit`** method requires an integer argument.

```
System.exit(0);
```

  - This argument is an *exit code* that is passed back to the operating system

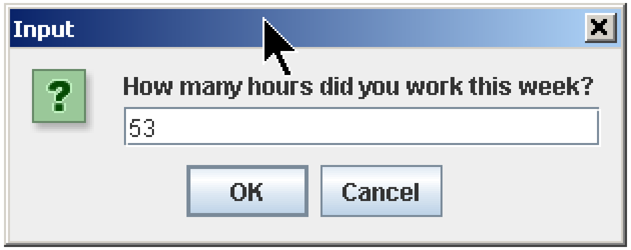
Lab (2)

- PayrollDialog
  - This program demonstrates using dialogs with `JOptionPane` with Input Dialog and Message Dialog



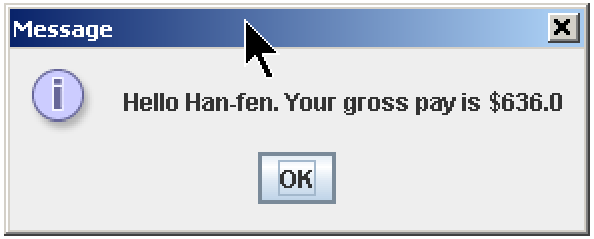
What is your name?

Han-fen



How many hours did you work this week?

53



Hello Han-fen. Your gross pay is $636.0

Converting a String to a Number

- Input dialogs always return the user's input as a **`String`**
- A **`String`** containing a number, such as “22”, can be converted to a numeric data type.

```
int number;
String str;
str = JOptionPane.showInputDialog("age?")
number = Integer.parseInt(str);
```

Parse Methods (1)

- Each of the numeric wrapper classes, has a method that converts a string to a number.
  - The **`Double`** class has a method that converts a string to a **`double`**, and
  - The **`Integer`** class has a method that converts a string to an **`int`**,
  - etc.
- These methods are known as *parse methods* because their names begin with the word “parse.”

Parse Methods (2)

```
// Store 1 in bVar.
byte bVar = Byte.parseByte("1");

// Store 2599 in iVar.
int iVar = Integer.parseInt("2599");

// Store 10 in sVar.
short sVar = Short.parseShort("10");

// Store 15908 in lVar.
long lVar = Long.parseLong("15908");

// Store 12.3 in fVar.
float fVar = Float.parseFloat("12.3");

// Store 7945.6 in dVar.
double dVar = Double.parseDouble("7945.6");
```

- The parse methods all throw a **`NumberFormatException`** if the **`String`** object does not represent a numeric value.

UNLV

`toString()` Methods

- Each of the numeric wrapper classes has a static **`toString`** method that converts a number to a string.
- The method accepts the number as its argument and returns a string representation of that number.

```
int i = 12;
double d = 14.95;
String str1 = Integer.toString(i);
String str2 = Double.toString(d);
```

`Character` Class

- The **`Character`** class allows a char data type to be *wrapped* in an object.
- The **`Character`** class provides methods that allow easy testing, processing, and conversion of character data.
  - These methods are very useful in verifying user input!
  - E.g.
    - Numbers,
    - A certain letter,
    - Remove space, etc.

Character Class Methods

Method	Description
boolean isDigit(char ch)	Returns true if the argument passed into ch is a digit from 0 through 9. Otherwise returns false.
boolean isLetter(char ch)	Returns true if the argument passed into ch is an alphabetic letter. Otherwise returns false.
boolean isLetterOrDigit(char ch)	Returns true if the character passed into ch contains a digit (0 through 9) or an alphabetic letter. Otherwise returns false.
boolean isLowerCase(char ch)	Returns true if the argument passed into ch is a lowercase letter. Otherwise returns false.
boolean isUpperCase(char ch)	Returns true if the argument passed into ch is an uppercase letter. Otherwise returns false.
boolean isSpaceChar(char ch)	Returns true if the argument passed into ch is a space character. Otherwise returns false.

Lab (3)

- CustomerNumber
  - In this program, the user will enter a customer number, and we’d like to verify whether the input follows the required format.

Searching Strings (1)

- The **`String`** class provides several methods that search for a string inside of a string.
- A *substring* is a string that is part of another string.
- Some of the substring searching methods provided by the **`String`** class:

```
boolean startsWith(String str)

boolean endsWith(String str)

boolean regionMatches(int start, String str, int
   start2, int n)

boolean regionMatches(boolean ignoreCase, int start,
                      String str, int start2, int n)
```

Searching Strings (2)

- The **`startsWith`** method determines whether a string begins with a specified substring.

```
String str = "Four score and seven years ago";
if (str.startsWith("Four"))
  System.out.println("The string starts with Four.");
else
  System.out.println("The string does not start with Four.");
```

  - **`str.startsWith("Four")`** returns true because **`str`** does begin with “Four”.

- **`startsWith`** is a case sensitive comparison.

Searching Strings (3)

- The **endsWith** method determines whether a string ends with a specified substring.

```
String str = "Four score and seven years ago";
if (str.endsWith("ago"))
  System.out.println("The string ends with ago.");
else
  System.out.println("The string does not end with ago.");
```

- The **endsWith** method also performs a case sensitive comparison.

Lab (4)

- PersonSearch.java
  - In this program, we’ll ask the user to enter a few characters, and the program will compare the input to the elements in the array.

Searching Strings (4)

- The **`String`** class provides methods that will if specified regions of two strings match.
  - **`regionMatches(int`** ***`start`***, **`String`** ***`str`***, **`int`** ***`start2`***, **`int`** ***`n`*****`)`**
    - returns true if the specified regions match or false if they don’t
    - Case sensitive comparison
  - **`regionMatches(boolean`** ***`ignoreCase`***, **`int`** ***`start`***, **`String`** ***`str`***, **`int`** ***`start2`***, **`int`** ***`n`*****`)`**
    - If ***`ignoreCase`*** is true, it performs case insensitive comparison

Searching Strings (5)

- The **`String`** class also provides methods that will locate the position of a substring.
  - **`indexOf`**
    - returns the first location of a substring or character in the calling **`String`** Object.
  - **`lastIndexOf`**
    - returns the last location of a substring or character in the calling **`String`** Object.

Searching Strings (6)

```
String str = "Four score and seven years ago";
int first, last;
first = str.indexOf('r');
last = str.lastIndexOf('r');
System.out.println("The letter r first appears at "
                    + "position " + first);
System.out.println("The letter r last appears at "
                   + "position " + last);
```

```
String str = "and a one and a two and a three";
int position;
System.out.println("The word and appears at the "
                   + "following locations.");

position = str.indexOf("and");
while (position != -1)
{
  System.out.println(position);
  position = str.indexOf("and", position + 1);
}
```

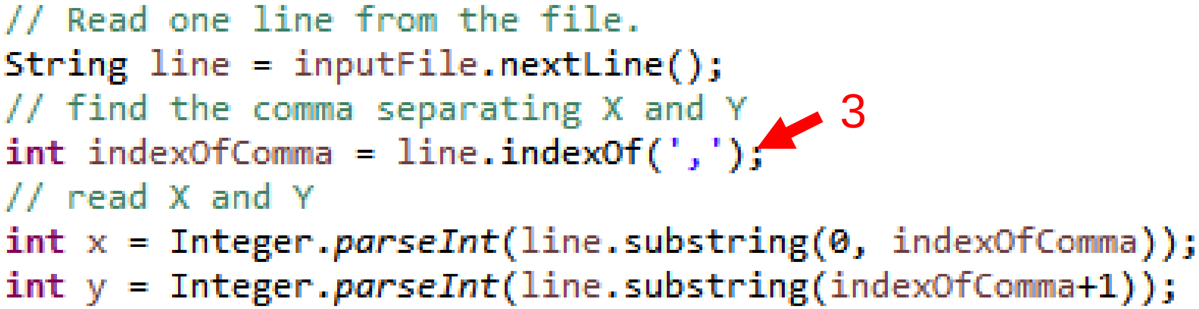
Extracting Substrings (1)

- The **`String`** class provides methods to extract substrings in a **`String`** object.
  - The **`substring`** method returns a substring beginning at a start location and an optional ending location.

```
String fullName = "Cynthia Susan Smith";
String lastName = fullName.substring(14);
System.out.println("The full name is "+ fullName);
System.out.println("The last name is "+ lastName);
```

Example of Substring and Parse Method

- In our previous lab example BombGame, we read the x and y coordinates from a file



```
// Read one line from the file.
String line = inputFile.nextLine();
// find the comma separating X and Y
int indexOfComma = line.indexOf(',');
// read X and Y
int x = Integer.parseInt(line.substring(0, indexOfComma));
int y = Integer.parseInt(line.substring(indexOfComma+1));
```

3

line.substring(0,3)

line.substring(4)

Data read from a file is a string. Convert the "356" and "87" to integer values

3	5	6	,	8	7
[0]	[1]	[2]	[3]	[4]	[5]

Extracting Characters to Arrays

- The **`String`** class provides methods to extract substrings in a **`String`** object and store them in **`char`** arrays.
  - **`getChars`**
    - Stores a substring in a **`char`** array
  - **`toCharArray`**
    - Returns the **`String`** object’s contents in an array of **`char`** values.

Lab (5)

- StringAnalyzer.java
  - In this program, the user will enter a string. The program will count the letters, digits and spaces in the string.