Chapter 3
Introduction to Object-Oriented Programming: Using Classes

Module Structure Chart

Main

Prepare File for Reading

Get Data

Print Data

Find Weighted Average

Print Weighted Average

Print Heading

Two Design Strategies

FUNCTIONAL DECOMPOSITION

OBJECT-ORIENTED DESIGN

Topics
- Class Basics and Benefits
- Creating Objects Using Constructors
- Calling Methods
- Using Object References
- Calling Static Methods and Using Static Class Variables
- Using Predefined Java Classes

Object-Oriented Programming
- Classes combine data and the methods (code) to manipulate the data
- Classes are a template used to create specific objects
- All Java programs consist of at least one class.
- Two types of classes
  - Application/Applet classes
  - Service classes
**Example**

- **Student class**
  - Data: name, year, and grade point average
  - Methods: store/get the value of each piece of data, promote to next year, etc.
- **Student Object: student1**
  - Data: Maria Gonzales, Sophomore, 3.5

**Some Terminology**

- **Object reference**: identifier of the object
- **Instantiating an object**: creating an object of a class
- **Instance of the class**: the object
- **Methods**: the code to manipulate the object data
- **Calling a method**: invoking a service for an object.

**Class Data**

- **Instance variables**: variables defined in the class and given values in the object
- **Fields**: instance variables and static variables (we'll define static later)
- **Members** of a class: the class's fields and methods
- Fields can be:
  - any primitive data type (*int, double, etc.*)
  - objects

**Encapsulation**

- Instance variables are usually declared to be *private*, which means users of the class must reference the data of an object by calling methods of the class.
- Thus the methods provide a protective shell around the data. We call this *encapsulation*.
- Benefit: the class methods can ensure that the object data is always valid.

**Naming Conventions**

- Class names: start with a capital letter
- Object references: start with a lowercase letter
- In both cases, internal words start with a capital letter
- Example: class: **Student**
  - objects: **student1**, **student2**

**Reusability**

- **Reuse**: class code is already written and tested, so you build a new application faster and it is more reliable

Example: A *Date* class could be used in a calendar program, appointment-scheduling program, online shopping program, etc.
How To Reuse A Class

• You don’t need to know how the class is written.
• You do need to know the application programming interface (API) of the class.
• The API is published and tells you:
  – How to create objects
  – What methods are available
  – How to call the methods

1. Declare an Object Reference

Syntax:
```java
ClassName objectReference;
```
or
```java
ClassName objectRef1, objectRef2…;
```
• Object reference holds address of object
• Example:
  ```java
  Date d1;
  ```

2. Instantiate an Object

• Objects MUST be instantiated before they can be used
• Call a constructor using `new` keyword
• Constructor has same name as class.
• Syntax:
  ```java
  objectReference =
  new ClassName( arg list );
  ```
• `Arg list` (argument list) is comma-separated list of initial values to assign to object data

Date Class API

**constructor**: special method that creates an object and assigns initial values to data

<table>
<thead>
<tr>
<th>Date Class Constructor Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date( )</td>
</tr>
<tr>
<td>Date( int mm, int dd, int yy )</td>
</tr>
</tbody>
</table>

Constructor

• **Constructor method** special method with the same name as the class that is used with `new` when a class is instantiated

```java
public class name {
    public name(String frst,String lst) {
        first = frst;
        last = lst;
    }
    Name name;
    name = new Name(“john”, “Dewey”);
}
```

Note: argument cannot be the same as field

Instantiation Examples

```java
Date independenceDay;
independenceDay = new Date( 7, 4, 1776 );
```
```java
Date graduationDate =
    new Date( 5, 15, 2008 );
```
```java
Date defaultDate = new Date( );
```
Example 3.1 Constructors.java

```java
public class Constructors {
    public static void main( String[] args ) {
        Date independenceDay;
        independenceDay = new Date( 7, 4, 1776 );
        Date graduationDate = new Date( 5, 15, 2008 );
        Date defaultDate = new Date();
    }
}
```

Objects After Instantiation

Method Classifications

- **Accessor** methods
  - `get...`
  - gives values of object data

- **Mutator** methods
  - `set...`
  - change values of object data

Date Class Methods

<table>
<thead>
<tr>
<th>Return value</th>
<th>Method name and argument list</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td><code>getMonth()</code></td>
<td>returns the value of month</td>
</tr>
<tr>
<td>int</td>
<td><code>getDay()</code></td>
<td>returns the value of day</td>
</tr>
<tr>
<td>int</td>
<td><code>getYear()</code></td>
<td>returns the value of year</td>
</tr>
<tr>
<td>void</td>
<td><code>setMonth(int mm)</code></td>
<td>sets the value of month to <code>mm</code></td>
</tr>
<tr>
<td>void</td>
<td><code>setDay(int dd)</code></td>
<td>sets the value of day to <code>dd</code></td>
</tr>
<tr>
<td>void</td>
<td><code>setYear(int yy)</code></td>
<td>sets the value of year to <code>yy</code></td>
</tr>
</tbody>
</table>

The Argument List in an API

- **Pairs of**
  - `dataType variableName`

- **Specify**
  - Order of arguments
  - Data type of each argument

- **Arguments can be:**
  - Any expression that evaluates to the specified data type
• When calling a method, include only expressions in your argument list. Including data types in your argument list will cause a compiler error.

• If the method takes no arguments, remember to include the empty parentheses after the method's name. The parentheses are required even if there are no arguments.

Void Methods

• **Void method** Does not return a value

  ```java
  System.out.print("Hello");
  System.out.println("Good bye");
  name.setName("CS", "201");
  ```

Value-Returning Methods

• **Value-returning method** Returns a value to the calling program

  ```java
  String first; String last;
  Name name;
  System.out.print("Enter first name: ");
  first = inData.readLine();
  System.out.print("Enter last name: ");
  last = inData.readLine();
  name.setName(first, last);
  ```

Value-returning example

```java
public String firstLastFormat()
{
    return first + " " + last;
}
```

Argument to print method is string returned from firstLastFormat method

Method Return Values

• Can be a primitive data type, class type, or `void`

• A **value-returning method**
  - Return value is not `void`
  - The method call is used in an expression. When the expression is evaluated, the return value of the method replaces the method call.

• Methods with a `void` return type
  - Have no value
  - Method call is complete statement (ends with ;)

Dot Notation

• Use when calling method to specify which object's data to use in the method

• Syntax:

  ```java
  objectReference.methodName( arg1, arg2, ... )
  ```

Note: no data types in method call; values only!
Example 3.2 Methods.java

```java
public class Methods {
    public static void main(String[] args) {
        Date independenceDay = new Date(7, 4, 1776);
        int independenceMonth = independenceDay.getMonth();
        System.out.println("Independence day is in month " + independenceMonth);
        Date graduationDate = new Date(5, 15, 2008);
        System.out.println("The current day for graduation is " + graduationDate.getDay());
        graduationDate.setDay(12);
        System.out.println("The revised day for graduation is " + graduationDate.getDay());
    }
}
```

Object Reference vs. Object Data

- Object references point to the location of object data.
- An object can have multiple object references pointing to it.
- Or an object can have no object references pointing to it. If so, the garbage collector will free the object's memory.
- See

Example 3.3 ObjectReferenceAssignment.java

```java
public class ObjectReferenceAssignment {
    public static void main(String[] args) {
        Date hireDate = new Date(2, 15, 2003);
        System.out.println("hireDate is " + hireDate.getMonth() + "/" + hireDate.getDay() + "/" + hireDate.getYear());
        Date promotionDate = new Date(9, 28, 2004);
        System.out.println("promotionDate is " + promotionDate.getMonth() + "/" + promotionDate.getDay() + "/" + promotionDate.getYear());
        promotionDate = hireDate;
        System.out.println("After assigning hireDate to promotionDate:");
        System.out.println("hireDate is " + hireDate.getMonth() + "/" + hireDate.getDay() + "/" + hireDate.getYear());
        System.out.println("promotionDate is " + promotionDate.getMonth() + "/" + promotionDate.getDay() + "/" + promotionDate.getYear());
    }
}
```

Two References to an Object

- After Example 3.3 runs, two object references point to the same object

null Object References

- An object reference can point to no object. In that case, the object reference has the value `null`
- Object references have the value `null` when they have been declared, but have not been used to instantiate an object.
- Attempting to use a `null` object reference causes a `NullPointerException` at runtime.
- See Example 3.5 NullReference2.java

Example 3.4 NullReference.java

```java
public class NullReference {
    public static void main(String[] args) {
        Date aDate;
    }
}
```
Example 3.5 NullReference2.java

```java
public class NullReference2 {
    public static void main(String[] args) {
        Date independenceDay = new Date(7, 4, 1776);
        System.out.println("The month of independenceDay is " + independenceDay.getMonth());
        independenceDay = null;
        // attempt to use object reference
        System.out.println("The month of independenceDay is " + independenceDay.getMonth());
    }
}
```

Date.java Class

```java
import java.awt.Graphics;
public class Date {
    private int month; private int day; private int year;
    public Date() {
        setDate(1, 1, 2000);
    }
    public Date(int mm, int dd, int yyyy) {
        setDate(mm, dd, yyyy);
    }
    /* accessor methods */
    public int getMonth() { return month; }
    public int getDay() { return day; }
    public int getYear() { return year; }
    /** mutator method */
    public void setMonth(int mm) {
        month = (mm >= 1 && mm <= 12 ? mm : 1);
    }
    public void setDay(int dd) {
        int[] validDays = {0, 31, 30, 31, 31, 30, 30, 31, 31, 30, 31, 30, 31};
        day = (dd >= 1 && dd <= validDays[month] ? dd : 1);
    }
    public void setYear(int yyyy) {
        year = yyyy;
    }
    public void setDate(int mm, int dd, int yyyy) {
        setMonth(mm);
        setDay(dd);
        setYear(yyyy);
    }
    public String toString() {
        return month + "/" + day + "/" + year;
    }
    public boolean equals(Date d) {
        if (month == d.month && day == d.day && year == d.year)
            return true;
        else
            return false;
    }
}
```

static Methods

- Also called class methods
- Can be called without instantiating an object
- Might provide some quick, one-time functionality, for example, popping up a dialog box
- In method API, keyword static precedes return type
  ```java
  static dataType methodName (arg1, arg2, ...);
  ```

Calling static Methods

- Use dot syntax with class name instead of object reference
- Syntax:
  ```java
  ClassName.methodName(args)
  ```
- Example:
  ```java
  int absValue = Math.abs(-9);
  ```
- Uses of class methods
  - Provide access to class variables without using an object

static Class Variables

- Syntax:
  ```java
  ClassName.staticVariable
  ```
- Example:
  ```java
  Color.BLUE
  ```
  BLUE is a static constant of the Color class.
Static Class Variables and Static Methods

class Counter {
private int value;
private static int numCounters = 0;
public Counter() {
    value = 0;
    numCounters++;
}
public static int getNumCounters() {
    return numCounters;
}
... System.out.println("Number of counters: "+ Counter.getNumCounters());
}

Class (static) vs. instance variables
– Instance variable: each instance has its own copy
– Class variable: the class has one copy for all instances

Can use instance variables
– In instance methods only

Can use class variables
– In instance methods
– In class methods

Using Java Predefined Classes

• Java Packages
• The String Class
• Using System.out
• Formatting Output
• The Math Class
• The Wrapper Classes
• Dialog Boxes
• Console Input Using the Scanner Class

Java Packages

• Classes are grouped in packages according to functionality

<table>
<thead>
<tr>
<th>Package</th>
<th>Categories of Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang</td>
<td>Basic functionality common to many programs, such as the String class and Math class</td>
</tr>
<tr>
<td>java.awt</td>
<td>Graphics classes for drawing and using colors</td>
</tr>
<tr>
<td>java.swing</td>
<td>User-interface components</td>
</tr>
<tr>
<td>java.text</td>
<td>Classes for formatting numeric output</td>
</tr>
<tr>
<td>java.util</td>
<td>The Scanner class and other miscellaneous classes</td>
</tr>
</tbody>
</table>

Java Predefined Classes

• Included in the Java SDK are more than 2,000 classes that can be used to add functionality to our programs

• APIs for Java classes are published on Sun Microsystems Web site: http://www.java.sun.com

• Also see Appendix F

Using a Class From a Package

• Classes in java.lang are automatically available to use

• Classes in other packages need to be "imported" using this syntax:
  ```java
  import package.ClassName;
  ```
  or
  ```java
  import package.*;
  ```

• Example
  ```java
  import java.text.DecimalFormat;
  ```
  or
  ```java
  import java.text.*;
  ```
The **String** Class

- Represents a sequence of characters

**String** constructors:

```java
String( String str )
    allocates a String object with the value of str, which can be String object or a String literal

String( )
    allocates an empty String
```

**String Concatenation Operators**

- `+` appends a String to another String. At least one operand must be a String
- `+=` shortcut String concatenation operator

- See Example 3.6 StringDemo.java

The **length** Method

<table>
<thead>
<tr>
<th>Return type</th>
<th>Method name and argument list</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>length( )</td>
</tr>
</tbody>
</table>

returns the number of characters in the String

**Example:**

```java
String hello = "Hello";
int len = hello.length( );
The value of len is 5
```

The **toUpperCase** and **toLowerCase** Methods

<table>
<thead>
<tr>
<th>Return type</th>
<th>Method name and argument list</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>toUpperCase( )</td>
</tr>
<tr>
<td>String</td>
<td>toLowerCase( )</td>
</tr>
</tbody>
</table>

returns a copy of the String will all letters upper case
returns a copy of the String will all letters lowercase

**Example:**

```java
String hello = "Hello";
hello = hello.toUpperCase( );
The value of hello is "HELLO"
```

The **indexOf** Methods

<table>
<thead>
<tr>
<th>Return type</th>
<th>Method name and argument list</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>indexOf( String searchString )</td>
</tr>
<tr>
<td>int</td>
<td>indexOf( char searchChar )</td>
</tr>
</tbody>
</table>

returns the index of the first character of searchString or -1 if not found
returns the index of the first character of searchChar or -1 if not found

The index of the first character of a String is 0.

**Example:**

```java
String hello = "Hello";
int index = hello.indexOf( 'e' );
The value of index is 1.
```

The **substring** Method

<table>
<thead>
<tr>
<th>Return type</th>
<th>Method name and argument list</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>substring( int startIndex, int endIndex )</td>
</tr>
</tbody>
</table>

returns a substring of the String object beginning at the character at index startIndex and ending at the character at index ( endIndex – 1 )

**Example:**

```java
String hello = "Hello";
String lo = hello.substring( 3, hello.length( )-1 );
The value of lo is 'lo'
```
• Specifying a negative start index or a start index past the last character of the String will generate a `StringIndexOutOfBoundsException`.
• Specifying a negative end index or an end index greater than the length of the String will also generate a `StringIndexOutOfBoundsException`.

System.out
• `System` is a class in java.lang package
• `out` is a a static constant field, which is an object of class `PrintStream`.
• `PrintStream` is a class in java.io package
• Since `out` is static we can refer to it using the class name `System.out`
• `PrintStream` Class has 2 methods for printing, `print` and `println` that accept any argument type and print to the standard java console.

Using `System.out`

<table>
<thead>
<tr>
<th>Return type</th>
<th>Method name and argument list</th>
</tr>
</thead>
</table>
| void        | `print(anyDataType argument)`  
|             | prints argument to the standard output device (by default, the Java console) |
| void        | `println(anyDataType argument)`  
|             | prints argument to the standard output device (Java console) followed by a newline character |

• Example:
  `System.out.print("The answer is ");`  
  `System.out.println(3);`

  Output is:  
  The answer is 3

The `toString` Method

<table>
<thead>
<tr>
<th>Return type</th>
<th>Method name and argument list</th>
</tr>
</thead>
</table>
| String      | `toString()`  
|             | converts the object data to a String for printing |

• All classes have a `toString` method which converts an object to string for printing

• See Example 3.7 `PrintDemo.java`

Formatting Numeric Output

• `NumberFormat` Class and the `DecimalFormat` Class allow you to specify the number of digits to print and add dollar signs and percent signs to your output

  - $5.25
  - 22%

• Both classes are in the `java.text` package

The `NumberFormat` Class

<table>
<thead>
<tr>
<th>Return type</th>
<th>Method name and argument list</th>
</tr>
</thead>
</table>
| `NumberFormat` | `getCurrencyInstance()`  
|             | static method that creates a format object for printing numbers as money |
| `NumberFormat` | `getPercentInstance()`  
|             | static method that creates a format object for printing percentages |
| `String` | `format(double number)`  
|             | returns a formatted String representation of number |

• See Example 3.8 `DemoNumberFormat.java`
### The `DecimalFormat` Class
- **Constructor:**
  ```java
  DecimalFormat( String pattern )
  ```
  instantiates a `DecimalFormat` object with the format specified by `pattern`

- **Pattern characters:**
  - `0` required digit
  - `#` optional digit, suppress if 0
  - `.` decimal point
  - `,` comma separator
  - `%` multiply by 100 and display a percent sign

- *See Example 3.9 DemoDecimalFormat*

### The Math Class Constants
- **Two static constants**
  - `PI` - the value of pi
  - `E` - the base of the natural logarithm

- **Example:**
  ```java
  System.out.println( Math.PI );
  System.out.println( Math.E );
  ```
  output is:
  ```
  3.141592653589793
  2.718281828459045
  ```

### Methods of the Math Class
- **All methods are static**

<table>
<thead>
<tr>
<th>Return type</th>
<th>Method name and argument list</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs( dataType arg )</code></td>
<td>returns the absolute value of the argument <code>arg</code>, which can be a <code>double</code>, <code>float</code>, <code>int</code> or <code>long</code></td>
</tr>
<tr>
<td><code>log( double a )</code></td>
<td>returns the natural logarithm (in base e) of its argument</td>
</tr>
<tr>
<td><code>sqrt( double a )</code></td>
<td>returns the positive square root of <code>a</code></td>
</tr>
<tr>
<td><code>pow( double base, double exp )</code></td>
<td>returns the value of base raised to the power of <code>exp</code></td>
</tr>
</tbody>
</table>

*See Examples 3.10 and 3.11*

### The Math `round` Method
- **Rounding rules:**
  - Any fractional part < .5 is rounded down
  - Any fractional part .5 and above is rounded up

*See Example 3.12 MathRounding.java*

### The Math `min/max` Methods
- **Find smallest of three numbers:**
  ```java
  int smaller = Math.min( num1, num2 );
  int smallest = Math.min( smaller, num3 );
  ```

*See Example 3.13 MathMinMaxMethods.java*

### The Math `random` Method
- **Generates a pseudorandom number** (appearing to be random, but mathematically calculated)
- **To generate a random integer between `a` and up to, but not including, `b`:**
  ```java
  int randomNum = a + (int)( Math.random( ) * ( b - a ) );
  ```

*See Example 3.14 MathRandomNumber.java*
The Wrapper Classes

• "wraps" the value of a primitive data type into an object

• Useful when methods require an object argument

• Also useful for converting Strings to an int or double

Wrapper Classes

<table>
<thead>
<tr>
<th>Primitive Data Type</th>
<th>Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>short</td>
<td>Short</td>
</tr>
<tr>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

Autoboxing and Unboxing

• **Autoboxing:**
  – Automatic conversion between a primitive type and a wrapper object when a primitive type is used where an object is expected
    Integer intObject = 42;

• **Unboxing**
  – Automatic conversion between a wrapper object and a primitive data type when a wrapper object is used where a primitive data type is expected
    int fortyTwo = intObject;

Integer and Double Methods

- **static Integer Methods**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Method Name and argument list</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>parseInt(String s)</td>
</tr>
<tr>
<td></td>
<td>returns the String s as an int</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th>Method Name and argument list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>valueOf(String s)</td>
</tr>
<tr>
<td></td>
<td>returns the String s as an Integer object</td>
</tr>
</tbody>
</table>

- **static Double Methods**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Method Name and argument list</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>parseDouble(String s)</td>
</tr>
<tr>
<td></td>
<td>returns the String s as a double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return value</th>
<th>Method Name and argument list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>valueOf(String s)</td>
</tr>
<tr>
<td></td>
<td>returns the String s as a Double object</td>
</tr>
</tbody>
</table>

See Example 3.15 DemoWrapper.java

Using Dialog Boxes

- **JOptionPane** class is in the javax.swing package

- **static** methods provided for input and output dialog boxes
  - showInputDialog for input
  - showMessageDialog for output

- For input dialog boxes, return value is a String, so numeric input needs to be converted (using parseInt or parseDouble)
### JOptionPane static Methods

<table>
<thead>
<tr>
<th>Return value</th>
<th>Method name and argument list</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td><code>showInputDialog( Component parent, Object prompt )</code> pops up an input dialog box, where prompt asks the user for input.</td>
</tr>
<tr>
<td>void</td>
<td><code>showMessageDialog( Component parent, Object message )</code> pops up an output dialog box with message displayed.</td>
</tr>
</tbody>
</table>

*See Examples 3.16 and 3.17*

---

### Input Using the Scanner Class

- Provides methods for reading `byte, short, int, long, float, double, and String` data types from the Java console.
- `Scanner` is in the `java.util` package.
- `Scanner` parses (separates) input into sequences of characters called tokens.
- By default, tokens are separated by standard white space characters (tab, space, newline, etc.).

---

### A Scanner Constructor

- `Scanner(InputStream source)` creates a `Scanner` object for reading from `source`.
  - If `source` is `System.in`, this instantiates a `Scanner` object for reading from the Java console.

*Example:*

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

---

### Scanner next… Methods

<table>
<thead>
<tr>
<th>Return type</th>
<th>Method name and argument list</th>
</tr>
</thead>
</table>
| `dataType`  | `nextDataType()` returns the next token in the input stream as a `dataType`. `dataType` can be `byte, int, short, long, float, double, or boolean`.
| `String`    | `next()` returns the next token in the input stream as a `String`.
| `String`    | `nextLine()` returns the remainder of the line as a `String`.

---

### Prompting the User

- Unlike dialog boxes, the `next…` methods do not prompt the user for an input value.
- Use `System.out.print` to print the prompt, then call the `next…` method.

*Example:*

```java
Scanner scan = new Scanner( System.in );
System.out.print( "Enter your age > " );
int age = scan.nextInt( );
```

*See Examples 3.18, 3.19*
• End your prompts with an indication that input is expected

• Include a trailing space for readability

System.in Object

Class BufferReader (returns String)

Class InputStreamReader (returns unicode characters)

Object InputStream (System.in)

Returns bytes

Input Streams

• Stream is flow of data
  – Reader at one end
  – Writer at the other end

• Stream generalizes input & output
  – Keyboard electronics different from disk
  – Input stream makes keyboard look like a disk

Input Streams: System.in

• System.in: the standard input stream
  – By default, reads characters from the keyboard

• Can use System.in many ways
  – Directly (low-level access)
  – Through layers of abstraction (high-level access)
Input Streams: Read Characters

• Can read characters from System.in with read()
  // Reads a single character from the keyboard and displays it
  class DemonstrateRead
  {
    public static void main(String[] args) throws java.io.IOException
    {
      char character;
      // Prompt for a character and read it
      System.out.print("Enter a character: ");
      System.out.flush();
      character = (char) System.in.read();
      // Display the character typed
      System.out.println("You typed "+ character);
    }
  }

Input Streams: Read Characters

• System.in.read() returns an integer
  – Usually cast to char
  *char character = (char) System.in.read();
  • System.in.read returns -1 if EOF detected
    – EOF = end of file (no more characters in stream)
  • Signaling EOF at keyboard
    – Control-Z (Windows), Control-D (Unix)
  • Example: Read all characters in stream until EOF
    int intChar = System.in.read();
    final int EOF=-1;
    while (intChar != EOF){
      // Convert to character
      char character = (char) intChar;
      System.out.println("Next character is "+ character);
      // Get next one
      intChar = System.in.read();
    }

Input Streams: Read Characters

• Example: Count digits, letters, other characters
  – Part 1: Setup
    // Reads text from the keyboard and displays the
    // number of digits, upper case letters, lower case
    // and other characters that the user typed.
    class CountCharacters
    {
      public static void main(String[] args)
      throws java.io.IOException
      {
        int nextValue, numUpperCase = 0, numLowerCase = 0,
        numDigits = 0, numOther = 0;
        char nextChar;
        // Display instructions
        System.out.println("Enter some text, terminate
        with EOF");
        // Read from the input stream, count
        // characters
        nextValue = System.in.read();
        while (nextValue != -1){
          nextChar = (char) nextValue;
          if (Character.isDigit(nextChar)) {
            numDigits ++;
          } else if (Character.isUpperCase(nextChar)) {
            numUpperCase ++;
          } else if (Character.isLowerCase(nextChar)) {
            numLowerCase ++;
          } else {
            numOther ++;
          }
          nextValue = System.in.read();
        }
      }
    }

Input Streams: Read Characters

• No String-reading methods in System.in
• To read strings from keyboard
  – First wrap System.in inside InputStreamReader object
    • InputStreamReader anInputStreamReader
      = new InputStreamReader(System.in);
  • read() returns char
    char character = (char) anInputStreamReader.read();
  • Example: Read all characters in stream until EOF
    int intChar = anInputStreamReader.read();
    final int EOF=-1;
    while (intChar != EOF){
      // Convert to character
      char character = (char) intChar;
      System.out.println("Next character is "+ character);
    // Get next one
      intChar = anInputStreamReader.read();
    }
Input Streams: Read Strings

- Next, wrap InputStreamReader object in BufferedReader object

```java
InputStreamReader anInputStreamReader = new InputStreamReader(System.in);
BufferedReader inStream = new BufferedReader(anInputStreamReader);
```

Can combine these two statements

```java
BufferedReader inStream = new BufferedReader(new InputStreamReader(System.in));
```

Skeleton for reading

```java
import java.io.*;
class ClassName {
    public static void main(String[] args) throws java.io.IOException {
        // Create a buffered input stream and attach it to standard input
        BufferedReader inStream = new BufferedReader(new InputStreamReader(System.in));
        ...}
}
```

Input Streams: Read Strings

- Methods in BufferedReader
  - `read()`: Use same as `System.in.read()`
  - `readLine()`: Returns complete line typed by user

- Example: Read user's name

```java
import java.io.*
class ReadInputAsString {
    public static void main(String[] args) throws java.io.IOException {
        String firstName, lastName;
        char middleInitial;
        // Create an input stream and attach it to the standard input stream
        BufferedReader inStream = new BufferedReader(new InputStreamReader(System.in));
        // Read a line from the user as a String
        System.out.print("Enter your first name: ");
        System.out.flush();
        firstName = inStream.readLine();
        // Read a character from the user
        System.out.print("Enter your middle initial and last name: ");
        System.out.flush();
        middleInitial = (char) inStream.read();
        // Read a line from the user as a String
        lastName = inStream.readLine();
        // Display the strings
        System.out.println();
        System.out.println("Your name is " + firstName + " " + middleInitial + ". " + lastName);
    }
}
```

Input Streams: Read Numbers

- To read a number from keyboard (int, double, ...)
  - Define a `NumberFormat` object
  - Define BufferedReader object
  - Use BufferedReader object to read response as a string
  - Use `NumberFormat` object to parse string into `Number` object
  - Convert `Number` object to primitive type

```java
import java.io.*;
class ReadInputAsNumber {
    public static void main(String[] args) throws java.io.IOException {
        // Define a NumberFormat object
        NumberFormat aNumberFormatter = NumberFormat.getInstance();
        // Define BufferedReader object
        BufferedReader inStream = new BufferedReader(new InputStreamReader(System.in));
        // Read response as a String
        System.out.print("Enter an integer: ");
        System.out.flush();
        String response = inStream.readLine();
        // Use NumberFormat object to parse string into Number object
        Number aNumberObject = aNumberFormatter.parse(response);
        // Convert Number object to primitive type
        int anIntNumber = aNumberObject.intValue();
    }
}
```
Input Streams: Read Numbers

- Can combine reading, parsing, conversion steps

```java
import java.io.*;
import java.text.NumberFormat;

class ReadAnInt2 {
    public static void main(String[] args)
        throws java.io.IOException, java.text.ParseException {
        // Create an input stream and attach it to the standard input stream
        BufferedReader inStream = new BufferedReader(new InputStreamReader(System.in));
        // Create a number formatter object
        NumberFormat aNumberFormatter = NumberFormat.getInstance();
        System.out.print("Enter an integer: ");
        // Read the response from the user, convert to Number then convert to int
        intNumber = aNumberFormatter.parse(inStream.readLine()).intValue();
        System.out.println("You typed ");
    }
}
```

Note
ParseException!