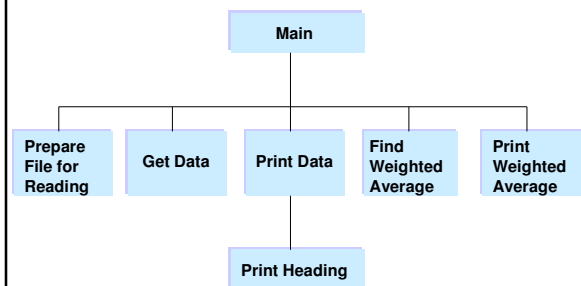


## Chapter 3

### Introduction to Object-Oriented Programming: Using Classes

## OO Programming

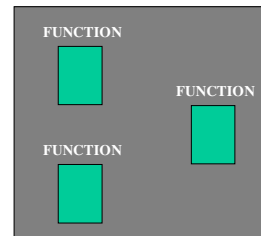
### Module Structure Chart



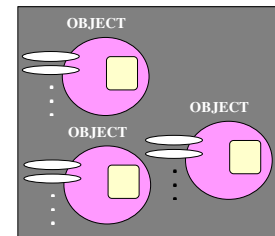
[CS201 Home Page](#)

### Two Design Strategies

#### FUNCTIONAL DECOMPOSITION



#### OBJECT-ORIENTED DESIGN



[CS201 Home Page](#)

### 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

[CS201 Home Page](#)

### 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

[CS201 Home Page](#)

## 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

[CS201 Home Page](#)

## 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.

[CS201 Home Page](#)

## 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

[CS201 Home Page](#)

## 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.

[CS201 Home Page](#)

## 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*

[CS201 Home Page](#)

## 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.

[CS201 Home Page](#)

## 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

[CS201 Home Page](#)

## 1. Declare an Object Reference

Syntax:

```
ClassName objectReference;  
or  
ClassName objectRef1, objectRef2...;
```

- Object reference holds address of object
- Example:

```
Date d1;
```

[CS201 Home Page](#)

## 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:

```
objectReference =  
    new ClassName( arg list );
```
- *Arg list* (argument list) is comma-separated list of initial values to assign to object data

[CS201 Home Page](#)

## Constructor

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

```
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*

[CS201 Home Page](#)

## Date Class API

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

### Date Class Constructor Summary

<code>Date( )</code> creates a <i>Date</i> object with initial month, day, and year values of 1, 1, 2000
<code>Date( int mm, int dd, int yy )</code> creates a <i>Date</i> object with initial month, day, and year values of <i>mm</i> , <i>dd</i> , and <i>yy</i>

[CS201 Home Page](#)

## Instantiation Examples

```
Date independenceDay;  
independenceDay = new Date( 7, 4,  
    1776 );  
  
Date graduationDate =  
    new Date( 5, 15, 2008 );  
  
Date defaultDate = new Date( );
```

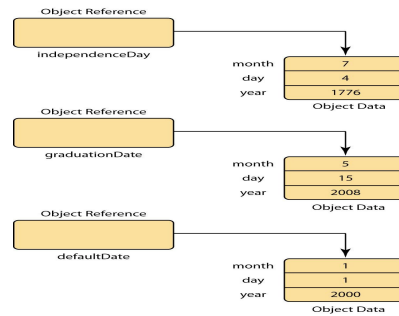
[Example Next Slide](#)

## Example 3.1 Constructors.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( );
    }
}
```

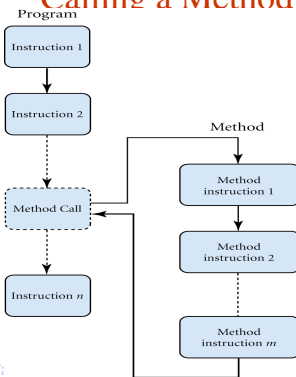
Figure 3.1 Next Slide

## Objects After Instantiation



CS201 H

## Calling a Method



CS201 Home Page

## Method Classifications

- **Accessor** methods
  - get...
  - gives values of object data
- **Mutator** methods
  - set...
  - change values of object data

CS201 Home Page

## Date Class Methods

Return value	Method name and argument list
int	<b>getMonth( )</b> returns the value of month
int	<b>getDay( )</b> returns the value of day
int	<b>getYear( )</b> returns the value of year
void	<b>setMonth( int mm )</b> sets the value of month to <i>mm</i>
void	<b>setDay( int dd )</b> sets the value of day to <i>dd</i>
void	<b>setYear( int yy )</b> sets the value of year to <i>yy</i>

CS201 Home Page

## 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

CS201 Home Page



- 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.

[CS201 Home Page](#)

## Void Methods

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

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

↑   ↑   ↑   ↑  
**object method arguments**

[CS201 Home Page](#)

## Value-Returning Methods

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

```
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);
```

[CS201 Home Page](#)

## Value-returning example

```
public String firstLastFormat()  
{  
    return first + " " + last;  
}  
System.out.print(name.firstLastFormat());
```

↑   ↑   ↑   ↑  
**object method object method**

Argument to **print** method is string returned from **firstLastFormat** method

[CS201 Home Page](#)

## 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 ;)

[CS201 Home Page](#)

## Dot Notation

- Use when calling method to specify which object's data to use in the method
- Syntax:

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

Note: *no data types in method call; values only!*

[Example Next Slide](#)

### Example 3.2 Methods.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() );
    }
}
```

CS201 Home Page

### 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 Next Slide

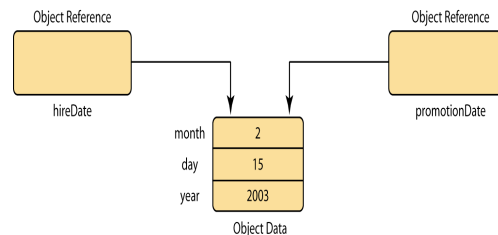
### Example 3.3 ObjectReferenceAssignment.java

```
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( "\nAfter 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() );
    }
}
```

CS201 Home Page

### Two References to an Object

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



CS201 Home Page

### 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 run time.
- See Example 3.5 *NullReference2.java*

CS201 Home Page

### Example 3.4 NullReference.java

```
public class NullReference
{
    public static void main( String[] args )
    {
        Date aDate;
        aDate.setMonth( 5 );
    }
}
```

CS201 Home Page

### Example 3.5 NullReference2.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() );
    }
}
```

[CS201 Home Page](#)

### Date.java Class

```
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 */
    int getMonth() { return month; }
    int getDay() { return day; }
    int getYear() { return year; }
    /* mutator method */
    public void setMonth( int mm ) {
        month = ( mm >= 1 && mm <= 12 ? mm : 1 );
    }
}
```

[CS201 Home Page](#)

### Date.java Class

```
public void setDay( int dd ) {
    int [] validDays = { 0, 31, 29, 31, 30, 31, 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;
}
}
```

[CS201 Home Page](#)

### 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  
`static dataType methodName (arg1, arg2, ...);`

[CS201 Home Page](#)

### Calling static Methods

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

[CS201 Home Page](#)

### static Class Variables

- Syntax:  
`ClassName.staticVariable`
- Example:  
`Color.BLUE`  
  
*BLUE* is a *static* constant of the *Color* class.

[CS201 Home Page](#)

## 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()); ←
}
```

[CS201 Home Page](#)

- 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

[CS201 Home Page](#)

## 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

[CS201 Home Page](#)

## 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

[CS201 Home Page](#)

## Java Packages

- Classes are grouped in **packages** according to functionality

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

[CS201 Home Page](#)

## 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:

```
import package.ClassName;
or
import package.*;
```

- Example

```
import java.text.DecimalFormat;
or
import java.text.*;
```

[CS201 Home Page](#)

## The *String* Class

- Represents a sequence of characters
- *String* constructors:

<code>String( String str )</code> allocates a <i>String</i> object with the value of <i>str</i> , which can be <i>String</i> object or a <i>String</i> literal
<code>String( )</code> allocates an empty <i>String</i>

[CS201 Home Page](#)

## *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*

[CS201 Home Page](#)

## The *length* Method

Return type	Method name and argument list
int	<code>length( )</code> returns the number of characters in the <i>String</i>

- Example:

```
String hello = "Hello";  
int len = hello.length( );
```

The value of *len* is 5

[CS201 Home Page](#)

## The *toUpperCase* and *toLowerCase* Methods

Return type	Method name and argument list
String	<code>toUpperCase( )</code> returns a copy of the <i>String</i> with all letters uppercase
String	<code>toLowerCase( )</code> returns a copy of the <i>String</i> with all letters lowercase

- Example:

```
String hello = "Hello";  
hello = hello.toUpperCase( );
```

The value of *hello* is "HELLO"

[CS201 Home Page](#)

## The *indexOf* Methods

Return type	Method name and argument list
int	<code>indexOf( String searchString )</code> returns the index of the first character of <i>searchString</i> or -1 if not found
int	<code>indexOf( char searchChar )</code> returns the index of the first character of <i>searchChar</i> or -1 if not found

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

- Example:

```
String hello = "Hello";  
int index = hello.indexOf( 'e' );
```

The value of *index* is 1.

[CS201 Home Page](#)

## The *substring* Method

Return type	Method name and argument list
String	<code>substring( int startIndex, int endIndex )</code> returns a substring of the <i>String</i> object beginning at the character at index <i>startIndex</i> and ending at the character at index ( <i>endIndex</i> - 1 )

- Example:

```
String hello = "Hello";  
String lo  
    = hello.substring( 3, hello.length( )-1 );
```

The value of *lo* is 'lo'

[CS201 Home Page](#)



- 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*

CS201 Home Page

## System.out

- **System** is a class in java.lang package
- **out** is 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.

CS201

## Using System.out

Return type	Method name and argument list
void	<b>print</b> ( anyDataType argument ) prints <i>argument</i> to the standard output device (by default, the Java console)
void	<b>println</b> ( anyDataType argument ) prints <i>argument</i> 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

CS201 Home Page

## The toString Method

Return type	Method name and argument list
String	<b>toString</b> ( ) converts the object data to a <i>String</i> for printing

- All classes have a *toString* method which converts an object to string for printing
- See Example 3.7 *PrintDemo.java*

CS201 Home Page

## 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

CS201 Home Page

## The NumberFormat Class

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

- See Example 3.8 *DemoNumberFormat.java*

CS201 Home Page

## The *DecimalFormat* Class

- **Constructor:**

```
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
- ,
- % multiply by 100 and display a percent sign

- See Example 3.9 *DemoDecimalFormat*

[CS201 Home Page](#)

## The *Math* Class Constants

- Two *static* constants

- PI - the value of pi
- E - the base of the natural logarithm

- **Example:**

```
System.out.println( Math.PI );
System.out.println( Math.E );
```

output is:

```
3.141592653589793
2.718281828459045
```

[CS201 Home Page](#)

## Methods of the *Math* Class

- All methods are *static*

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

- See Examples 3.10 and 3.11

[CS201 Home Page](#)

## The *Math* *round* Method

Return type	Method name and argument list
long	<b>round</b> ( double a ) returns the closest integer to its argument <i>a</i>

- **Rounding rules:**

- Any fractional part < .5 is rounded down
- Any fractional part .5 and above is rounded up

- See Example 3.12 *MathRounding.java*

[CS201 Home Page](#)

## The *Math* *min*/*max* Methods

Return type	Method name and argument list
dataTypeOfArgs	<b>min</b> ( dataType a, dataType b ) returns the smaller of the two arguments. The arguments can be <i>doubles</i> , <i>floats</i> , <i>ints</i> , or <i>longs</i> .
dataTypeOfArgs	<b>max</b> ( dataType a, dataType b ) returns the larger of the two arguments. The arguments can be <i>doubles</i> , <i>floats</i> , <i>ints</i> , or <i>longs</i> .

- Find smallest of three numbers:

```
int smaller = Math.min( num1, num2 );
int smallest = Math.min( smaller, num3 );
```

- 

- See Example 3.13 *MathMinMaxMethods.java*

[CS201 Home Page](#)

## The *Math* *random* Method

Return type	Method name and argument list
double	<b>random</b> ( ) returns a random number greater than or equal to 0 and less than 1

- 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*:

```
int randNum = a
+ (int)( Math.random() * ( b - a ) );
```

- See Example 3.14 *MathRandomNumber.java*

[CS201 Home Page](#)

## 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*

[CS201 Home Page](#)

## Wrapper Classes

Primitive Data Type	Wrapper Class
<i>double</i>	<i>Double</i>
<i>float</i>	<i>Float</i>
<i>long</i>	<i>Long</i>
<i>int</i>	<i>Integer</i>
<i>short</i>	<i>Short</i>
<i>byte</i>	<i>Byte</i>
<i>char</i>	<i>Character</i>
<i>boolean</i>	<i>Boolean</i>

[CS201 Home Page](#)

## 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;
```

[CS201 Home Page](#)

## Integer and Double Methods

- *static Integer Methods*

Return value	Method Name and argument list
int	<b>parseInt</b> ( String s ) returns the <i>String s</i> as an <i>int</i>
Integer	<b>valueOf</b> ( String s ) returns the <i>String s</i> as an <i>Integer</i> object

- *static Double Methods*

Return value	Method Name and argument list
double	<b>parseDouble</b> ( String s ) returns the <i>String s</i> as a <i>double</i>
Double	<b>valueOf</b> ( String s ) returns the <i>String s</i> as a <i>Double</i> object

*See Example 3.15 DemoWrapper.java*

[CS201 Home Page](#)

## Input Data

Dialog Box  
Java console (System.in)  
GUI (chapter 12)  
File (chapters 6, 11)

## 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`)

[CS201 Home Page](#)

## JOptionPane static Methods

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

- See Examples 3.16 and 3.17

[CS201 Home Page](#)



- Provide the user with clear prompts for input.
- Prompts should use words the user understands and should describe the data requested and any restrictions on valid input values.

- Example:

```
Enter your first and last name
or
Enter an integer between 0 and 10
```

[CS201 Home Page](#)

## 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.)

[CS201 Home Page](#)

## 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:

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

[CS201 Home Page](#)

## Scanner next... Methods

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

[CS201 Home Page](#)

## 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:

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

- 

See Examples 3.18, 3.19

[CS201 Home Page](#)



- End your prompts with an indication that input is expected
- Include a trailing space for readability

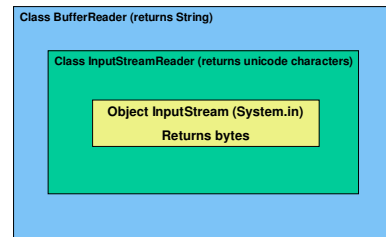
[CS201 Home Page](#)

## Backup

## Java Console

System.in  
Scanner

## System.in Object



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

[CS201 Home Page](#)

## Input Streams

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

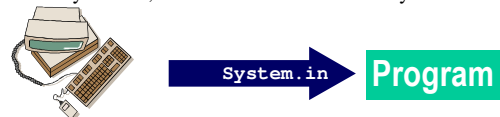


- Stream generalizes input & output
  - Keyboard electronics different from disk
  - Input stream makes keyboard look like a disk

[CS201 Home Page](#)

## 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)

[CS201 Home Page](#)

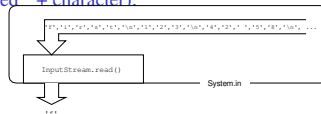
## Input Streams: Read Characters

- 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();
        System.out.println("You typed " + character);
    }
}
    
```



CS201 Home Page

## 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();
}
            
```

CS201 Home Page

## 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
// letters,
// 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");
    }
}
    
```

CS201 Home Page

## Input Streams: Read Characters

### -Part 2: Read and count

```

// Read from the input stream, count
// characters
//until no more characters in the input stream
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();
}
    
```

CS201 Home Page

## Input Streams: Read Characters

### - Part 3: Display results

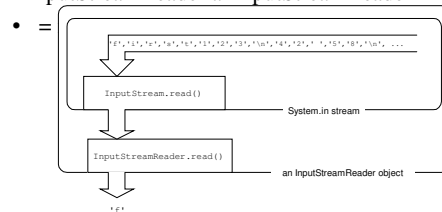
```

// Display results
System.out.println();
System.out.println();
System.out.println("Number of digits:"
    + numDigits);
System.out.println("Number of upper case letters: "
    + numUpperCase);
System.out.println("Number of lower case letters: "
    + numLowerCase);
System.out.println("Number of other characters: "
    + numOther);
}
}
    
```

CS201 Home Page

## Input Streams: Read Strings

- No String-reading methods in System.in
- To read strings from keyboard
  - First wrap System.in inside InputStreamReader object
    - InputStreamReader anInputStreamReader

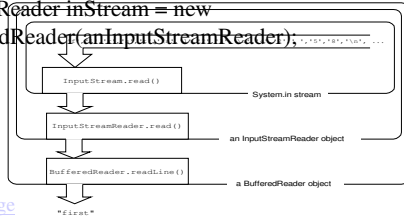


CS201 Home Page

## Input Streams: Read Strings

- Next, wrap InputStreamReader object in BufferedReader object

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



CS201 Home Page

## Input Streams: Read Strings

Can combine these two statements

```
BufferedReader inStream
= new BufferedReader(new InputStreamReader(System.in));
InputStreamReader & BufferedReader in java.io.*
Must import java.io.*;
```

Skeleton for reading

```
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));
...
}
```

CS201 Home Page

## 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

```
// Reads a user's first name, middle initial,
// and last name. Demonstrates use of InputStreamReader,
// BufferedReader and the readLine() method.
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));
```

CS201 Home Page

## Input Streams: Read Strings

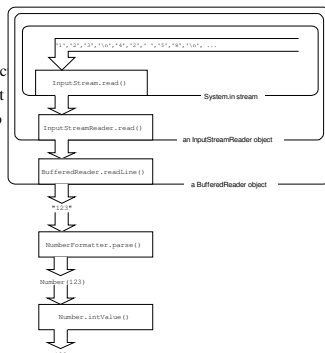
- Example: continued

```
// 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);
```

CS201 Home Page

## 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



CS201 Home Page

## Input Streams: Read Numbers

- Code to read a number from keyboard (int, double, ...)
  - 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 intNumber = aNumberObject.intValue();
```

CS201 Home Page

## Input Streams: Read Numbers

- Can combine reading, parsing, conversion steps

```
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
    int intNumber
        = aNumberFormatter.parse(inStream.readLine()).intValue();
    System.out.println("You typed " + intNumber);
    }
}
```

Note  
ParseException!

[CS201 Home Page](#)