COP 3330: Object-Oriented Programming
Summer 2011

Basic Java

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3330/sum2011

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida

The Anatomy of a Java Program

- A Java application program contains the following basic components:
  - Comments
  - Reserved Words
  - Modifiers
  - Statements
  - Blocks
  - Classes
  - Methods
  - The `main` method (note: Java applets do not have a main method)

Java Comments

- Comments are designed to enhance the readability of source code.
- There are three styles of comments in Java:
  - Line comments begin with // and consist of a single line only.
  - Block comments begin with /* and end with */ and can cover many lines of commenting. Convention also puts an * in the leftmost position of every line in the comment.
  - Javadoc comments begin with /** and end with */. They are used for documenting classes, data, and methods and can be extracted into an XHTML file using the JDK javadoc command. We'll deal much more with this type of comment later.

Java Comments

A `javadoc` comment

A block comment

A line (in line) comment

HeatIndexCalculator. | Rectangle.java | Greetings.java

```
/** This is a javadoc style comment
 *  Greetings class generates a Welcome to Java message
 *
 * @author Mark Llewellyn  May 23, 2011
 * No known bugs
 *
 */
public class Greetings {
    /* This is a block style Java comment.
     * Notice that it goes on for several lines
     * Most IDEs, including Eclipse will automatically generate
     * the leading * symbol for each line as you type.
     */

    public static void main (String[] args){
        //This is a single line comment in Java
        System.out.println("Greetings from the world of Java!");
    }//end main method
}//end class Greetings
```


Java File Layout Conventions

- Sun's layout conventions for Java source files suggest that you include the following components in the order given:
  - A block comment including the name of the file, the date, and any copyright information.
  - An optional package declaration and any include statements.
  - The public class or interface declaration.
  - Any nonpublic class or interface declarations.
- Within each class declaration, the class components (comments, fields, constructors, and methods) should be laid out in the following order:
  - A comment block containing class implementation details. These comments include any information that is not appropriate for javadoc comments, such as class invariants that are implementation specific.
  - Static fields, ordered in decreasing accessibility (public fields first, the protected, package, and finally private).
  - Instance fields, ordered similarly.
  - Constructors.
  - Methods, ordered by functionality.

Reserved Words in Java

- Reserved words or keywords, are words that have a specific meaning to the compiler and cannot be uses for any other purposes in a Java program.
- Note that Java is a case-sensitive language, which means that while `public` is a reserved word `Public` is not. However, from a readability perspective, it is best to avoid a reserved word in any form except that for which it was intended. (Note: `goto` and `const` are C++ reserved words not presently used in Java.)

Reserved Words in Java

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Modifiers in Java

- Java uses certain reserved words called modifiers that specify the properties of the data, methods, and classes and how they can be used.

Modifier	Applicable to					Explanation
	Class	Constructor	Method	Data	Block	
`(default)`	yes	yes	yes	yes	yes	A class, constructor, method , or data field is visible in this package. Default has no access modifier keyword.
`public`	yes	yes	yes	yes	no	A class, constructor, method , or data field is visible to all the programs in any package.
`private`	no	yes	yes	yes	no	A constructor, method, or data field is only visible in this class.
`protected`	no	yes	yes	yes	no	A constructor, method, or data field is visible in this package and in subclasses of this class in any package.
`static`	no	no	yes	yes	yes	Define a class method, or a class data field, or a static initialization block.
`final`	yes	no	yes	yes	no	A final class cannot be extended. A final method cannot be modified in a subclass. A final data field is a constant.
`abstract`	yes	no	yes	no	no	An abstract class must be extended. An abstract method must be implemented in a concrete subclass.

Statements in Java

- A statement represents an action or sequence of actions.
- Every statement in Java ends with a semi-colon.

HeatIndexCalculator. | Rectangle.java | Greetings.java | »5

```java
/** This is a javadoc style comment
 *  Greetings class generates a Welcome to Java message
 *
 * @author Mark Llewellyn  May 23, 2011
 * No known bugs
 *
 */
public class Greetings {
    /* This is a block style Java comment.
     * Notice that it goes on for several lines
     * Most IDEs, including Eclipse will automatically generate
     * the leading * symbol for each line as you type.
     */

    public static void main (String[] args){
        //This is a single line comment in Java
        System.out.println("Greetings from the world of Java!");
    }//end main method
}//end class Greetings
```

Not a Java statement

A Java statement

Blocks in Java

- In Java, each block, begins with an opening brace ({) and ends with a closing brace (}).
- Every class has a class block that groups the data and methods of the class.
- Every method has a method block the groups the statements in the method.
- Blocks can be nested, placing one block inside of another block.

HeatIndexCalculator. | Rectangle.java | Greetings.java

```
/** This is a javadoc style comment
 *  Greetings class generates a Welcome to Java message
 *
 * @author Mark Llewellyn  May 23, 2011
 * No known bugs
 *
 */
public class Greetings {
    /* This is a block style Java comment.
     * Notice that it goes on for several lines
     * Most IDEs, including Eclipse will automatically generate
     * the leading * symbol for each line as you type.
     */

    public static void main (String[] args){
        //This is a single line comment in Java
        System.out.println("Greetings from the world of Java!");
    }//end main method
}//end class Greetings
```

A class block opening brace

A method block

Corresponding closing brace for the class block

Classes in Java

- The class is the essential Java construct. To develop software in Java, you must understand classes and be able to write and use them. We've seen an introduction to classes so far, in that classes define the objects which are the agents of action in a Java program.
- A Java program is defined by one or more classes.
- A class is the Java mechanism for allowing the programmer to specify a new type of object and instantiate instances (objects) of the class.
- A class allows an information type to be designed and implemented only once and then reused as often as needed without having to reanalyze and rejustify the implementation.

Methods in Java

- A method in Java is the specification of a behavior that an object (an instance) of the class may exhibit.
- As we mentioned earlier, a method encapsulates an action or a service that an object of the class can perform when requested.

```
public class Person
  {
     private String name;
     public Person (String who)
     {
       this.name = who;
     }

     public String getName()
     {
       return name;
     }

  }
```

```
//create two Person objects
Person aGirl = new Person("Debi");
Person anotherGirl = new Person("Eva");
String girl1 = aGirl.getName();
//girl1 now has value of "Debi"
String girl2 = anotherGirl.getName();
//girl2 now has value of "Eva"
```


The `main` Method in Java

- Every Java application must have a user-declared `main` method where the program execution begins. (Note: Java applets do not have a main method.)
- The `main` method is always a `public static void` method.
- The `main` method has the following form (either one works):

```
public static void main (String[] args)
{
   //statements;
}
```

```
public static void main (String args[])
{
   //statements;
}
```


The `main` Method in Java

HeatIndexCalculator. | Greetings.java | Echo.java | 6

```java
/** Echo class simply prints or echos the arguments passed
 *  to it via the command line.
 *
 * @author Mark Llewellyn May 23, 2011
 *
 *  No known bugs
 */
public class Echo {

    public static void main (String[] args){
        for (int i = 0; i < args.length; i++){
            System.out.print(args[i] + "  ");
        }//end for loop
        System.out.println();
    }//end main method
}//end class Echo
```

This program simply echos the command line arguments passed to the main method when execution begin.

Java - Basic Java/src/Echo.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

Basic Java
- src
  - (default package)
    - Echo.java
    - Greetings.java

HeatIndexCalculator. | Rectangle.java | Greetings.java | Echo.java

```
/** Echo class simply prints or echos the arguments passed
 *  to it via the command line.
 *
 * @author Mark Llewellyn May 23, 2011
 *
 *  No known bugs
/
blic class Echo {

  public static void main (String[] args){
      for (int i = 0; i < args.length; i++){
          System.out.print(args[i] + "  ");
      }//end for loop
      System.out.println();
  }//end main method
/end class Echo
```

Click Run tab, the click Run Configurations in the drop-down list that appears.

You can also click the Run icon down arrow to get to the same place.

Problems | Javadoc | Declaration | Console

<terminated> Greetings [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 23, 2011 1:31:30 PM)

```
Greetings from the world of Java!
```

Writable | Smart Insert | 4 : 39

Run Configurations

Create, manage, and run configurations

Run a Java application

type filter text

- Java Applet
- Java Application
  - FirstProgram
  - Greetings
  - HeatIndexCalculator
  - Person
  - Rectangle
  - TestRectangle
  - TestRectangle2
- JUnit
- Task Context Test

Filter matched 11 of 11 items

Name: Greetings

Main | Arguments | JRE | Classpath | Source | Environment | Common

Project:

Basic Java — Browse...

Main class:

Echo — Search...

- Include system libraries when searching for a main class
- Include inherited mains when searching for a main class
- Stop in main

Apply | Revert

Run | Close

Click the Arguments tab in this dialog box

Run Configurations

Create, manage, and run configurations

Run a Java application

type filter text

- Java Applet
- Java Application
  - FirstProgram
  - Greetings
  - HeatIndexCalculator
  - Person
  - Rectangle
  - TestRectangle
  - TestRectangle2
- JUnit
- Task Context Test

Name: Greetings

Main | Arguments | JRE | Classpath | Source | Environment | Common

Program arguments:

Debi
Kristi
Vanessa
Tammi

Variables...

VM arguments:

Variables...

g directory:

ult: ${workspace_loc:Basic Java}

er:

Workspace... File System... Variables...

Apply Revert

Run Close

Enter your command line arguments in this window.

Then click Apply.

Then click Run

Java - Basic Java/src/Echo.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

HeatIndexCalculator. | Greetings.java | Echo.java

```
/** Echo class simply prints or echos the arguments passed
 *  to it via the command line.
 *
 * @author Mark Llewellyn May 23, 2011
 *
 *  No known bugs
 */
```

Problems | Javadoc | Declaration | Console

<terminated> Greetings [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 23, 2011 1:

```
Debi   Kristi   Vanessa   Tammi
```

Execution window showing echoing of the command line arguments

Java - Basic Java/src/GreetingsUsingADialogBox.java

File Edit Source Refactor Navigate Search P

HeatIndexCalculator. Greetings.java

```
/** GreetingsUsingADialogBox prints a welcome message using a dialog box
 *
 * @author Mark Llewellyn May 23, 2011
 *
 * No known bugs
 *
 */
import javax.swing.JOptionPane;

public class GreetingsUsingADialogBox {

    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "Welcome to Java!",
                "Greetings From Java", JOptionPane.INFORMATION_MESSAGE);
    }//end main method
```

This program uses the Java class `JOptionPane`. Java's predefined classes are grouped into packages. The `JOptionPane` class is in the `javax.swing` package. In the previous example, we did not need to import the System class because it is in the `java.lang` package and all classes in this package are implicitly imported into every Java program.

The showMessageDialog method is a static method. Static methods are invoked by using the class name followed by the dot operator and the method name with any arguments.

If you replace `JOptionPane` here (both places) with `javax.swing.JOptionPane` you would not need the import statement at the top. Import statements allow a shorthand notation to be used to referencing a class within the package that is imported. Try it!

Java - Basic Java/src/GreetingsUsingADialogBox.java - Eclipse

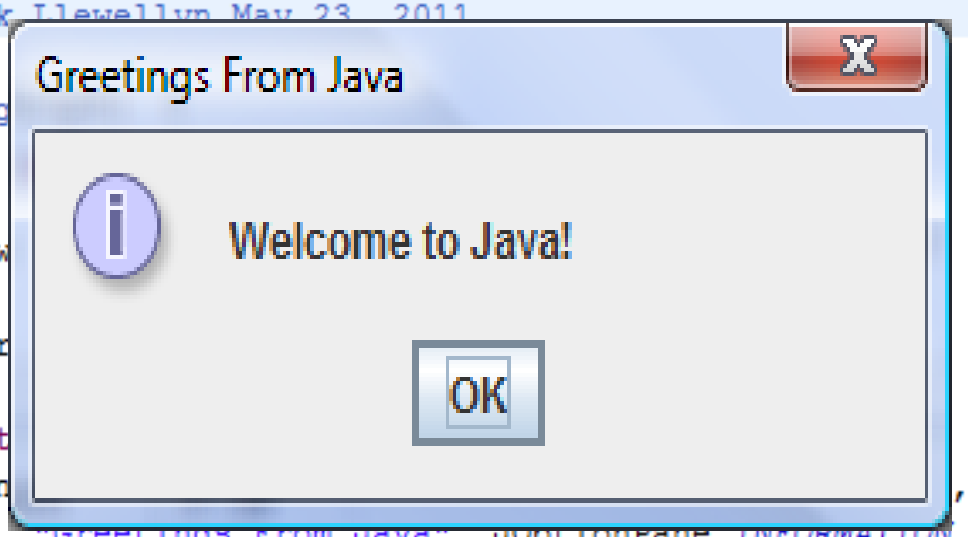
File Edit Source Refactor Navigate Search Project Run Window Help

HeatIndexCalculator. | Greetings.java | Echo.java | GreetingsUsingADialo

```
/** GreetingsUsingADialogBox prints a welcome message using a dialog box
 *
 * @author Mark Llewellyn May 23, 2011
 *
 * No known bug
 *
 */
import javax.sw

public class Gr

    public stat
        JOption
                "Greetings from Java", JOptionPane.INFORMATION_MESSAGE);
    }//end main method
```



Writable | Smart Insert | 3 : 39

Identifiers in Java

- Identifiers are used in Java (as in other programming languages) to name programming entities such as variables, constants, methods, class, and packages.
- The rules for naming identifiers in Java are:
  - An identifier is a sequence of characters that consists of letters, digits, underscores (_), and dollar signs ($).
  - An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
  - An identifier cannot be a reserved word (see page 7 for list of reserved words in Java).
  - An identifier cannot be the words `true`, `false`, or `null`.
  - An identifier can be of any length.
- Java is case-sensitive, so `X` and `x` are different identifiers.

Identifier Conventions in Java

- While identifier names should be as descriptive as possible, there are other style/convention guidelines that good programmers will follow to enhance the readability and maintainability of their code.
- The naming conventions for naming variables, methods, and classes are:
  - Use lowercase letters for variables and methods. If a name consists of several words, concatenate them into one word, making the first word lowercase and capitalizing the first letter of each subsequent word. For example, `radius, getName`, `showInputDialog`.
  - Capitalize the first letter of each word in a class name. For example, `ComputeArea, JOptionPane, ThisIsANewClass`.
  - Capitalize every letter in a constant, and use underscores between words. For example, PI, MAX_VALUE.

Variables in Java

- Variables are used for representing data of a certain type.
- To use a variable, you declare it by telling the compiler the name of the variables as well as what type of data it represents. This is called a variable declaration. Declaring a variable tells the compiler to allocated the appropriate memory space for the variable based on its data type.
- There are only two types in Java, primitive types and object types.
- There are eight primitive types in Java:
  - Integer types are: byte, short (2 bytes), int (4 bytes) , long (8 bytes)
  - Real number types are: float (typically 6 place accuracy) and double (typically 15 place accuracy)
  - Character type: char
  - Logical type: boolean

Numeric Data Types in Java

Type	Range	Storage Size
`byte`	-2^7 (-128) to 2^7-1 (+127)	8-bit signed
`short`	-2^{15} (-32768) to 2^{15}-1 (+32767)	16-bit signed
`int`	-2^{31} (-2147483648) to 2^{31}-1 (+2147483647)	32-bit signed
`long`	-2^{63} (-9223372036854775808) to 2^{63}-1 (+9223372046854775807)	64-bit signed
`float`	Negative range: -3.4028235E+38 to 1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754 standard
`double`	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754 standard

Declaring Variables in Java

- The syntax for declaring a variable in Java is:

```
datatype  variableName;
```

or

```
datatype  variable1, variable2,…, variablen;
```

- Some examples are:

```
int x;            //declare x to be an integer variable
double radius;    //declare radius to be a double variable
char a;           //declare a to be a character variable
```

- Variable of the same type can be declared together and are separated by commas.

```
int x,y,z;  //declare x, y, and z to be integer variables
```


Assignment Statements and Variables

- After a variable is declared, you can assign a value to it by using an assignment statement. In Java, the equal sign (=) is used as the assignment operator.
- The syntax for an assignment statement in Java is:

```
variable = expression;
```

- An expression represents a computation involving values, variables, and operators that together evaluates to a value. If the expression is legal, it must evaluate to the type of the variable to which the value is being assigned.
- Some examples:

```
int x = 1;      //this is a declaration and assignment in one step
double radius = 1.0;  //assign 1.0 to radius
 x = y + 1;  //assign to x the sum of y and 1
 s = s + PI;  //assignment involving variable on both sides of =
```


Constants

- While the value of a variable may change during the execution of a program, the value of a constant cannot change (that's why its called a constant!).
- A constant must be declared and initialized in the same statement. A constant is defined in Java by using the keyword `final`.
- The syntax for a constant definition is:

```
final datatype CONSTANT_NAME = value;
```

- Java convention capitalizes every letter in a constant.

Numeric Operations

Java Operator	Meaning	Example	Result
+	Addition	34 + 1	35
–	Subtraction	34.0 – 1.0	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder (modulo division)	20 % 3	2

Modulo division can be quite useful. For example, any even number % 2 is always 0, and any odd number % 2 is always 1. So this is a simple way to determine if a number is odd or even. Suppose that today is Saturday, you and your friend are going to meet in 10 days. What day is in 10 days?

Saturday is the 6th day of the week

(6 + 10) % 7 = 16 % 7 = 2, thus you will meet on a Tuesday.

You will meet in 10 days. There are 7 days in a week Tuesday is the 2nd day of the week

Shorthand Operations

Java Operator	Meaning	Example	Result
+=	Addition assignment	x += 8	x = x + 8
−=	Subtraction assignment	x –= 4.0	x = x – 4.0
*=	Multiplication assignment	x *= 2	x = x * 2
/=	Division assignment	x /= b	x = x / b
%=	Remainder assignment	x %= 5	x = x % 5

Java Operator	Meaning	Description
`++var`	preincrement	`var` is incremented by 1, then the new value of `var` is returned.
`var++`	postincrement	`var` is returned (old value) then incremented by 1.
`--var`	predecrement	`var` is decremented by 1, then the new value of `var` is returned.
`var--`	postdecrement	`var` is returned (old value) then decremented by 1.

Numeric Type Conversions

- Sometimes it is necessary to mix numeric values of different types in a computation.
- Java automatically converts numeric types in an expression according to the following rules:
  1. If one of the operands is `double`, the other is converted into a `double`.
  2. Otherwise, if one of the operands is a `float`, the other is converted into a `float`.
  3. Otherwise, if one of the operands is `long`, the other is converted into a `long`.
  4. Otherwise, both operands are converted into an `int`.

Numeric Type Conversions

- You can always assign a value to a numeric variable whose type supports a wider range of values . This is called a widening conversion or widening a type. For example, you can assign a long value to a float variable. Java performs widening conversions implicitly.
- In Java. you cannot assign a value to a variable of a type with a smaller range of values (a narrowing conversion or narrowing a type) unless you use explicit type casting.
- Casting is an operation that converts a value of one data type into a value of another data type.

Numeric Type Conversions

- The syntax for casting is to place the target type in parentheses, followed by the variable or the value to be cast.

```
float f = (float) 10.1;
int I = (int) f;
```

- Casting does not change the variable being cast.

Character and String Data Types

- Java supports Unicode which by today's standard is a 16-bit character code with a set of supplementary characters. Unicode contains just over a million different characters.
- A 16-bit Unicode takes two bytes (1 byte = 8 bits), In Java, a Unicode character is preceded by a `\u` and is expressed as 4 hexadecimal digits. Unicode runs from `\u0000` to `\uFFFF`.
- For example, the Unicode for the Greek letters α, β, and γ are, `\u03b1`, `\u03b2`, and `\u03b3`.

Character and String Data Types

Java - Basic Java/src/UniCodeExample.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

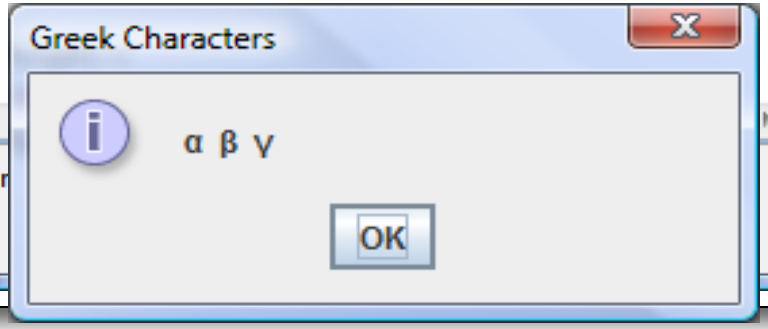
Greetings.java | Echo.java | GreetingsUsingADialo | *UniCodeExample.java

```
import javax.swing.JOptionPane;

/** UniCodeExample illustrates how to generate Unicode characters into
 *  Java program output.
 *
 * @author Mark Llewellyn May 23, 2011
 *
 * No known bugs
 */
public class UniCodeExample {

    public static void main(String[] args){
        JOptionPane.showMessageDialog(null, "\u03b1  \u03b2  \u03b3",
                "Greek Characters", JOptionPane.INFORMATION_MESSAGE);
    }//end main method
}//end class UniCodeExample
```

Writable | Smart Inser



Character and String Data Types

- Most computers use ASCII, which is a 7-bit encoding scheme for representing all uppercase and lowercase letters, digits, punctuation marks, and control characters. Unicode encompasses the entire ASCII code, with \u0000 to \u007F corresponding to the 128 ASCII characters ($2^7 = 128$).
- You can use ASCII characters as well as Unicode characters in a Java program.
- For example, the following two statements are equivalent in Java:

```
char letter = 'A';

char letter = '\u0041';  //character A's Unicode is 41
```


Character and String Data Types

- The character data type `char`, is used to represent a single character. A character literal is enclosed in single quotation marks.

```
char letter = 'A';

char numChar = '4';
```

- A string literal is enclosed in double quotation marks, So "A" is a string, and 'A' is a character.

Character and String Data Types

- The increment and decrement operators also apply to variables of the `char` type.

```
char ch = 'a';
System.out.println(++ch);  //prints character b
```

- The `char` type only represents one character. To represent a string of character, use the data type called `String`. String is actually a predefined class in the Java library, just like the System class and the JOptionPane class.

- The String type is not a primitive type, it is a reference type (an object).

Character and String Data Types

```
//Three string concatenated
String message1 = "Welcome" + " to" + " Java";
String message2 = "Welcome " + "to " + "Java";
//String Chapter is concatenated with number 2
Sting s = "Chapter"+2;  //s becomes Chapter2
//String Supplement is concatenated with character B
String s1 = "Supplement" + 'B';  //s1 becomes SupplementB
//if neither operand is a string, (+) adds two numbers
//prefix and postfix operations also works with strings
message1 += " and Java is fun"; //message1 is now "Welcome to Java and Java is fun"
// if i = 1 and j = 2
System.out.println("i + j is " + i + j);  //output is "i+j is 12"
//to force the evaluation of i+j, encloses the operation in parentheses
System.out.println("i + j is " + (i + j)); //output is "i+j is 3"
```


Casting Between `char` and Numeric Types

- A `char` can be cast into any numeric type and vice versa.
- When an integer is cast into a `char`, only its lower sixteen bits of data are used, the other part is simply ignored.

```
char c = (char)0XAB0041;
  //the lower 16 bit hex code 41 is assigned to c
System.out.println(c);  //c is the character A
```


Casting Between `char` and Numeric Types

- When an floating-point value is cast into a `char`, the integral part of the floating-point value is cast into a `char`.

```
char t = (char)65.25;
  //decimal 65 is assigned to t
System.out.println(t);  //t is the character A
```


Console Input Using the `Scanner` Class

- While there are several ways to enter data into a Java program while it is executing, one simple way is to use the `Scanner` class.
- Java uses `System.out` to refer to the standard output device (default is your terminal screen), and `System.in` to refer to the standard input device (default is your keyboard).
- To perform console output, you simply use the `println` method to display either a primitive value or a string to the screen. (Note: `print` and `println` are identical except that `println` moves the cursor to the next line after displaying the string.)
- Console input is not directly supported in Java, but you can use the `Scanner` class to create an object to read input from `System.in` as follows:

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


Console Input Using the `Scanner` Class

Method	Description
`nextByte()`	Reads an integer of the `byte` type
`nextShort()`	Reads an integer of the `short` type
`nextInt()`	Reads an integer of the `int` type
`nextLong()`	Reads an integer of the `long` type
`nextFloat()`	Reads a number of the `float` type
`nextDouble()`	Read a number of the `double` type
`next()`	Reads a string that ends before a whitespace. A whitespace character is `' '`, `'\t'`, `'\f'`, `'\r'`, or `'\n'`.
`nextLine()`	Reads a line of characters (i.e., a string ending with a line separator)

Methods In `Scanner` Class

Echo.java | GreetingsUsingADialo | UniCodeExample.java | *TestScanner.java

```java
import java.util.Scanner;  //Scanner is in java.util

public class TestScanner {
/**
 * @param args
 *
 */
  public static void main(String[] args) {
    //Create a Scanner object
    Scanner input = new Scanner(System.in);

    //Prompt user to enter an integer
    System.out.print("Enter an integer: ");
    int intValue = input.nextInt();
    System.out.println("You entered the integer: " + intValue);

    //Prompt the user to enter a double value
    System.out.print("Enter a double value: ");
    double doubleValue = input.nextDouble();
    System.out.println("You entered the double value: " + doubleValue);

    //Prompt the user to enter a string
    System.out.print("Enter a string without a space: ");
    String stringValue = input.next();
    System.out.println("You entered the string: " + stringValue);
  }//end main method
}//end class TestScanner
```


Java - Basic Java/src/TestScanner.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Console

<terminated> TestScanner [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 23, 2011 1:52:49 PM)

```
Enter an integer: 44
You entered the integer: 44
Enter a double value: 56.78
You entered the double value: 56.78
Enter a string without a space: Hello!
You entered the string: Hello!
```


Getting Input From Dialog Boxes

- We've already seen the `JOptionPane` class at work in a previous example (see page 18). We used this class to display a `showMessageDialog` box. The `JOptionPane` class also has a method `showInputDialog` that can be used to get input for a program at runtime.
- While the showInputDialog method can be used in several different ways, for the time being we'll only need to know two different ways to invoke this method.

Getting Input From Dialog Boxes

- One way is using a statement like:

```
String astring =
JOptionPane.showInputDialog(null, x, y,
JOptionPane.QUESTION_MESSAGE));
```

where `x` is a string for the prompting message, and `y` is a string for the title of the input dialog box.

- The other way is using a statement like:

```
JOptionPane.showInputDialog(x);
```

where `x` is a string for the prompting message.

Getting Input From Dialog Boxes

- The input returned from an input dialog box is a string. If you enter a numeric value such as 123, it returns '123'. You must convert a string into a number to obtain the input as a number.
- To convert a string into an `int`, use the `parseInt` method in the `Integer` class as follows:

```
int intValue = Integer.parseInt(intString);
```

where `intString` is a numeric string such as '123'.

- To convert a string into an `double`, use the `parseDouble` method in the `Double` class as follows:

```
doublent doubleValue = Double.parseDouble(doubleString);
```

where `doubleString` is a numeric string such as '123.45'.

- The `Integer` and `Double` classes are both included in the `java.lang` class and are automatically imported.

GreetingsUsingADialo | UniCodeExample.java | TestScanner.java | *InputDialogBoxExamp

```java
/** InputDialogBoxExample illustrates entering input from a
 *  dialog box.
 *
 * @author Mark Llewellyn May 23, 2011
 * No known bugs
 */
import javax.swing.JOptionPane;

public class InputDialogBoxExample {
/**
 * @param args
 */
    public static void main (String[] args){
        String input = JOptionPane.showInputDialog(null,
                "Enter a message", "Input Dialog Example",
                JOptionPane.QUESTION_MESSAGE);
        System.out.println("The value entered in the dialog box was: " +
                  input);
    }//end main method
}//end class InputDialogBoxExample
```

Java - Basic Java/src/InputDialogBoxExample.java - Eclipse

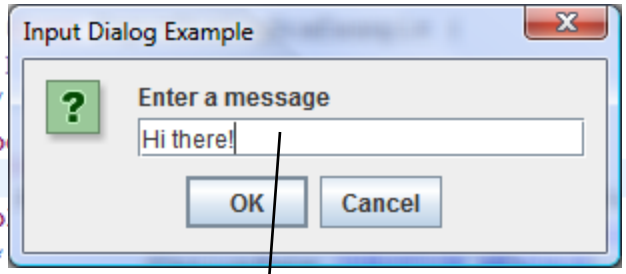
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- Basic Java
  - src
    - (default package)
      - Echo.java
      - Greetings.java
      - GreetingsUsingADialogBox.
      - InputDialogBoxExample.jav
      - TestScanner.java
      - UniCodeExample.java
  - JRE System Library [JavaSE-1.6]
- Day Two
- My First Java Programs
- Program One
  - src
    - (default package)
      - HeatIndexCalculator.java
  - JRE System Library [JavaSE-1.6]

UniCodeExample.java | TestScanner.java | InputDialogBoxExampl

```
/** InputDialogBoxExample illustrates entering input from a
 *  dialog box.
 *
 *
 *
 */
imp
pub
/**
 * @param args
 */
    public static void main (String[] args){
```



Problems | Javadoc | Declaration | Console

<terminated> InputDialogBoxExample [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (May 23, 2(

```
The value entered in the dialog box was: Hi there!
```

Writable | Smart Insert | 8 : 1