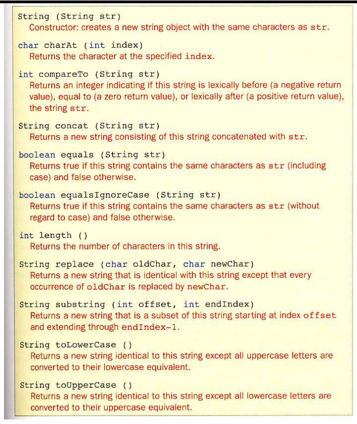
Formatting Output & Enumerated Types & Wrapper Classes

September 8, 2006

ComS 207: Programming I (in Java)
Iowa State University, FALL 2006
Instructor: Alexander Stoytchev

Quick review of last lecture

String Class



`String (String str)`
Constructor: creates a new string object with the same characters as `str`.

`char charAt (int index)`
Returns the character at the specified `index`.

`int compareTo (String str)`
Returns an integer indicating if this string is lexically before (a negative return value), equal to (a zero return value), or lexically after (a positive return value), the string `str`.

`String concat (String str)`
Returns a new string consisting of this string concatenated with `str`.

`boolean equals (String str)`
Returns true if this string contains the same characters as `str` (including case) and false otherwise.

`boolean equalsIgnoreCase (String str)`
Returns true if this string contains the same characters as `str` (without regard to case) and false otherwise.

`int length ()`
Returns the number of characters in this string.

`String replace (char oldChar, char newChar)`
Returns a new string that is identical with this string except that every occurrence of `oldChar` is replaced by `newChar`.

`String substring (int offset, int endIndex)`
Returns a new string that is a subset of this string starting at index `offset` and extending through `endIndex-1`.

`String toLowerCase ()`
Returns a new string identical to this string except all uppercase letters are converted to their lowercase equivalent.

`String toUpperCase ()`
Returns a new string identical to this string except all lowercase letters are converted to their uppercase equivalent.

The String Class

- **Because strings are so common, we don't have to use the new operator to create a `String` object**

```
title = "Java Software Solutions";
```

- **This is special syntax that works <u>only</u> for strings**
- **Each string literal (enclosed in double quotes) represents a `String` object**

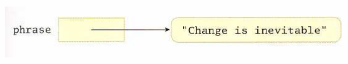
String Methods

- **Once a `String` object has been created, neither its value nor its length can be changed**
- **Thus we say that an object of the `String` class is *immutable***
- **However, several methods of the `String` class return new `String` objects that are modified versions of the original**
- **See the list of `String` methods on page 119 and in Appendix M**

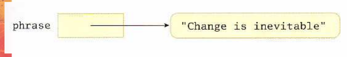
String Indexes

- **It is occasionally helpful to refer to a particular character within a string**
- **This can be done by specifying the character's numeric *index***
- **The indexes begin at zero in each string**
- **In the string `"Hello"`, the character `'H'` is at index 0 and the `'o'` is at index 4**
- **See StringMutation.java (page 120)**

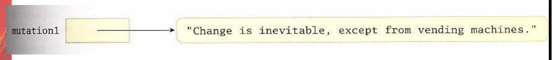
String Mutations Example



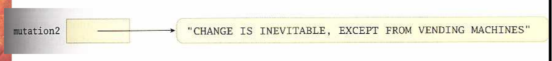
String Mutations Example



mutation1=phrase.concat(", except from vending machines.");



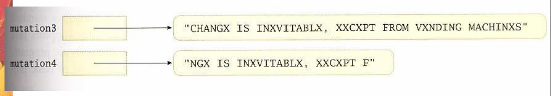
mutation2= mutation1.toUpperCase();

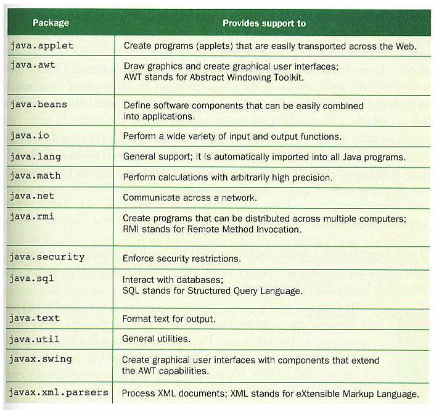


String Mutations Example

mutation3= mutation2.replace('E', 'X');

mutation4= mutation3.substring(3, 30);


mutation3
"CHANGX IS INXVITABLX, XXCXPT FROM VXNDING MACHINXS"
mutation4
"NGX IS INXVITABLX, XXCXPT F"



Package	Provides support to
`java.applet`	Create programs (applets) that are easily transported across the Web.
`java.awt`	Draw graphics and create graphical user interfaces; AWT stands for Abstract Windowing Toolkit.
`java.beans`	Define software components that can be easily combined into applications.
`java.io`	Perform a wide variety of input and output functions.
`java.lang`	General support; it is automatically imported into all Java programs.
`java.math`	Perform calculations with arbitrarily high precision.
`java.net`	Communicate across a network.
`java.rmi`	Create programs that can be distributed across multiple computers; RMI stands for Remote Method Invocation.
`java.security`	Enforce security restrictions.
`java.sql`	Interact with databases; SQL stands for Structured Query Language.
`java.text`	Format text for output.
`java.util`	General utilities.
`javax.swing`	Create graphical user interfaces with components that extend the AWT capabilities.
`javax.xml.parsers`	Process XML documents; XML stands for eXtensible Markup Language.

Class Libraries

- **A *class library* is a collection of classes that we can use when developing programs**
- **The *Java standard class library* is part of any Java development environment**
- **Its classes are not part of the Java language per se, but we rely on them heavily**
- **Various classes we've already used (`System` , `Scanner`, `String`) are part of the Java standard class library**
- **Other class libraries can be obtained through third party vendors, or you can create them yourself**

The import Declaration

- **When you want to use a class from a package, you could use its *fully qualified name***

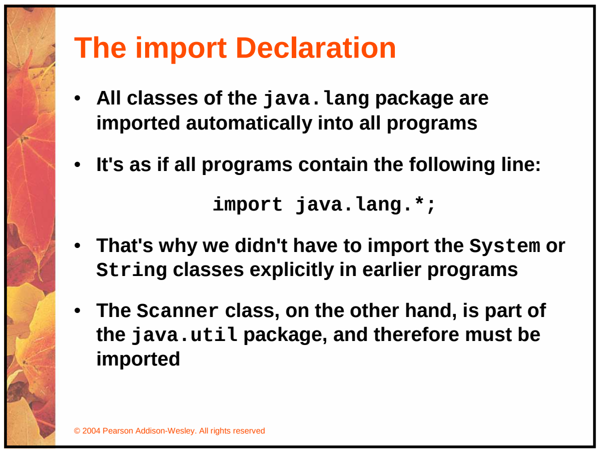
```
java.util.Scanner
```

- **Or you can *import* the class, and then use just the class name**

```
import java.util.Scanner;
```

- **To import all classes in a particular package, you can use the * wildcard character**

```
import java.util.*;
```

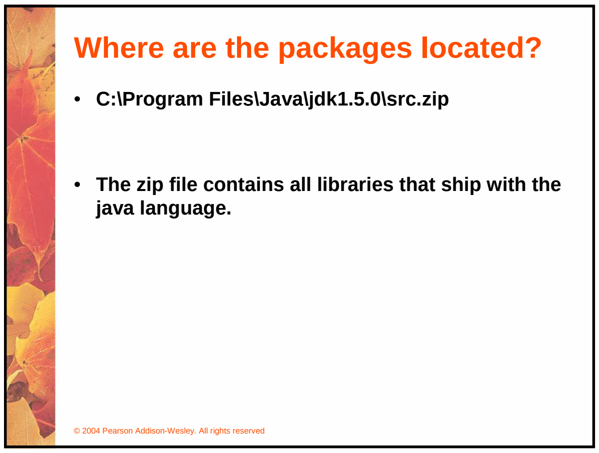


The import Declaration

- All classes of the `java.lang` package are imported automatically into all programs
- It's as if all programs contain the following line:

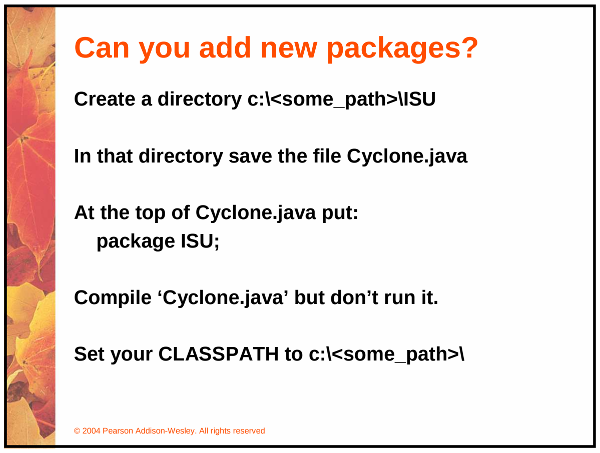
```
import java.lang.*;
```

- That's why we didn't have to import the `System` or `String` classes explicitly in earlier programs
- The `Scanner` class, on the other hand, is part of the `java.util` package, and therefore must be imported



Where are the packages located?

- C:\Program Files\Java\jdk1.5.0\src.zip
- The zip file contains all libraries that ship with the java language.



Can you add new packages?

Create a directory c:\<some_path>\ISU

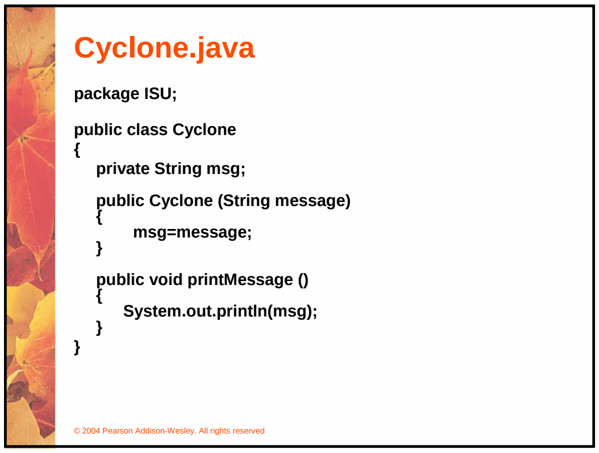
In that directory save the file Cyclone.java

At the top of Cyclone.java put:

```
package ISU;
```

Compile ‘Cyclone.java’ but don’t run it.

Set your CLASSPATH to c:\<some_path>\



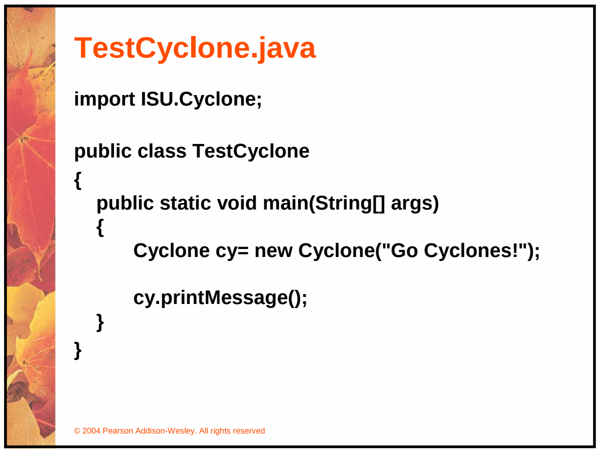
Cyclone.java

```
package ISU;

public class Cyclone
{
    private String msg;

    public Cyclone (String message)
    {
        msg=message;
    }

    public void printMessage ()
    {
        System.out.println(msg);
    }
}
```

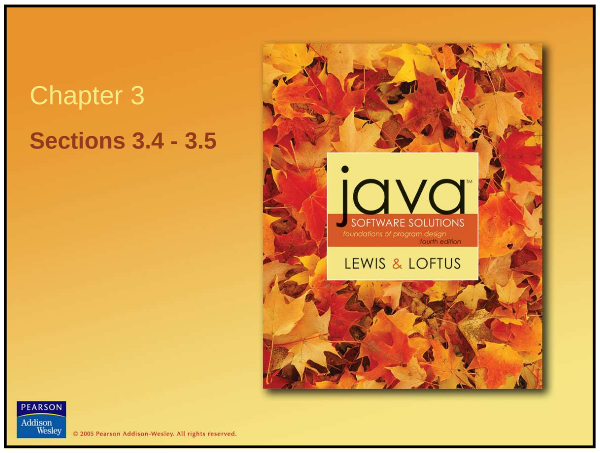


TestCyclone.java

```
import ISU.Cyclone;

public class TestCyclone
{
    public static void main(String[] args)
    {
        Cyclone cy= new Cyclone("Go Cyclones!");

        cy.printMessage();
    }
}
```



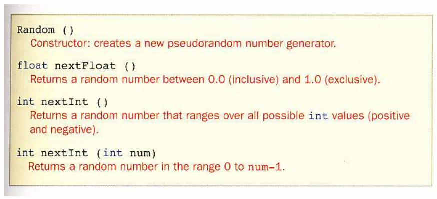
Chapter 3

Sections 3.4 - 3.5

The Random Class

- The `Random` class is part of the `java.util` package
- It provides methods that generate pseudorandom numbers
- A `Random` object performs complicated calculations based on a *seed value* to produce a stream of seemingly random values

Methods in The Random Class



```
Random ()
   Constructor: creates a new pseudorandom number generator.

float nextFloat ()
   Returns a random number between 0.0 (inclusive) and 1.0 (exclusive).

int nextInt ()
   Returns a random number that ranges over all possible int values (positive
   and negative).

int nextInt (int num)
   Returns a random number in the range 0 to num-1.
```

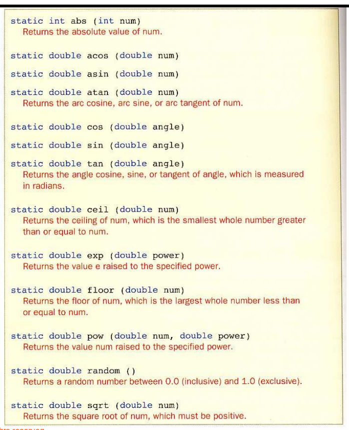
Random Numbers Example

- See RandomNumbers.java (page 126)

The Math Class

- The `Math` class is part of the `java.lang` package
- The `Math` class contains methods that perform various mathematical functions
- These include:
  - absolute value
  - square root
  - exponentiation
  - trigonometric functions

Math Class



```
static int abs (int num)
   Returns the absolute value of num.

static double acos (double num)

static double asin (double num)

static double atan (double num)
   Returns the arc cosine, arc sine, or arc tangent of num.

static double cos (double angle)

static double sin (double angle)

static double tan (double angle)
   Returns the angle cosine, sine, or tangent of angle, which is measured
   in radians.

static double ceil (double num)
   Returns the ceiling of num, which is the smallest whole number greater
   than or equal to num.

static double exp (double power)
   Returns the value e raised to the specified power.

static double floor (double num)
   Returns the floor of num, which is the largest whole number less than
   or equal to num.

static double pow (double num, double power)
   Returns the value num raised to the specified power.

static double random ()
   Returns a random number between 0.0 (inclusive) and 1.0 (exclusive).

static double sqrt (double num)
   Returns the square root of num, which must be positive.
```

The Math Class

- The methods of the `Math` class are *static methods* (also called *class methods*)
- Static methods can be invoked through the class name – no object of the `Math` class is needed

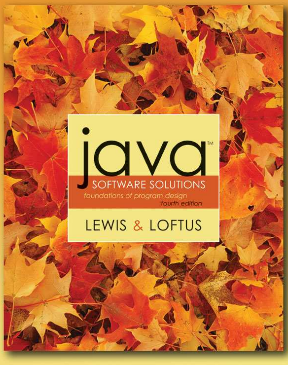
```
value = Math.cos(90) + Math.sqrt(delta);
```

- See Quadratic.java (page 129)
- We discuss static methods further in Chapter 6

Run examples from the book

Chapter 3

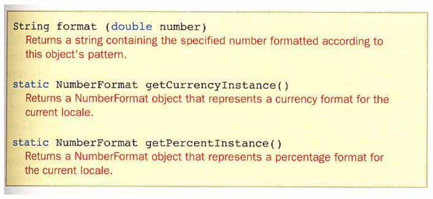
Sections 3.6 - 3.8



Formatting Output

- Two new classes
  - DecimalFormat
  - NumberFormat

Methods in NumberFormat Class



```
String format (double number)
```
Returns a string containing the specified number formatted according to this object's pattern.

```
static NumberFormat getCurrencyInstance()
```
Returns a NumberFormat object that represents a currency format for the current locale.

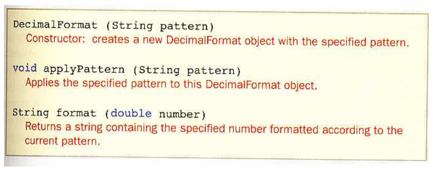
```
static NumberFormat getPercentInstance()
```
Returns a NumberFormat object that represents a percentage format for the current locale.

NumberFormat Example

```
double dollars=5.994;
NumberFormat fmt = NumberFormat.getCurrencyInstance();
System.out.println ( “Price = “ + fmt.format(dollars) );

RESULT:
Price = $5.99
```

Methods in DecimalFormat Class



```
DecimalFormat (String pattern)
```
Constructor: creates a new DecimalFormat object with the specified pattern.

```
void applyPattern (String pattern)
```
Applies the specified pattern to this DecimalFormat object.

```
String format (double number)
```
Returns a string containing the specified number formatted according to the current pattern.

DecimalFormat Example

```
double miles =  .5395;

DecimalFormat fmt = new DecimalFormat(“0.###”);
System.out.println ( “Miles = “ + fmt.format(miles) );

RESULT:
~~Miles = 0.540~~

Miles = 0.54
```

The printf Method

- Provided as a courtesy to C programmers
- System.out.printf(“ID: %5d\tName: %s”, id, name);

The printf convention

- %d print an int argument in decimal
- %ld print a long int argument in decimal
- %c print a character
- %s print a string
- %f print a float or double argument
- %e same as %f, but use exponential notation
- %g use %e or %f, whichever is better
- %o print an int argument in octal (base 8)
- %x print an int argument in hexadecimal (base 16)
- %% print a single %

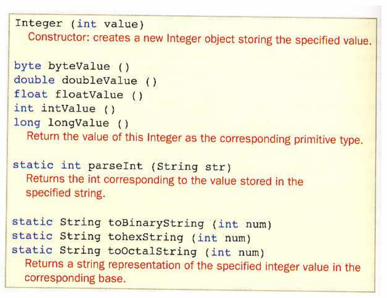
[From: www.eskimo.com/~scs/cclass/notes/sx6a.html]

Wrapper Classes

- The `java.lang` package contains *wrapper classes* that correspond to each primitive type:

Primitive Type	Wrapper Class
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`
`void`	`Void`

Integer Class



```
Integer (int value)
   Constructor: creates a new Integer object storing the specified value.

byte byteValue ()
double doubleValue ()
float floatValue ()
int intValue ()
long longValue ()
   Return the value of this Integer as the corresponding primitive type.

static int parseInt (String str)
   Returns the int corresponding to the value stored in the
   specified string.

static String toBinaryString (int num)
static String tohexString (int num)
static String toOctalString (int num)
   Returns a string representation of the specified integer value in the
   corresponding base.
```

Wrapper Classes

- The following declaration creates an `Integer` object which represents the integer 40 as an object

  `Integer age = new Integer(40);`

- An object of a wrapper class can be used in any situation where a primitive value will not suffice
- For example, some objects serve as containers of other objects
- Primitive values could not be stored in such containers, but wrapper objects could be

Wrapper Classes

- Wrapper classes also contain static methods that help manage the associated type
- For example, the `Integer` class contains a method to convert an integer stored in a `String` to an `int` value:

```
num = Integer.parseInt(str);
```

- The wrapper classes often contain useful constants as well
- For example, the `Integer` class contains `MIN_VALUE` and `MAX_VALUE` which hold the smallest and largest `int` values

Autoboxing

- *Autoboxing* is the automatic conversion of a primitive value to a corresponding wrapper object:

```
Integer obj;
int num = 42;
obj = num;
```

- The assignment creates the appropriate `Integer` object
- The reverse conversion (called *unboxing*) also occurs automatically as needed

Autoboxing Examples

```
Integer obj1;
int num1 = 69;
obj1 = num1;   // automatically creates an
               //integer object



Integer obj2= new Integer(69);
int num2;
num2 = obj2;   // automatically extracts
               //the int value
```

THE END