



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Basic Java Programming

Brighten Your Way And Raise You Up.



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 1

Getting Started

Brighten Your Way And Raise You Up.

- ◆ **Set up Java development environment**
- ◆ **Understand Java features**
- ◆ **Understand functions of Java virtual machine**
- ◆ **Describe the concept of garbage collection**
- ◆ **Enumerate the ways in which Java platform implements code security**
- ◆ **Define class, package and applications**
- ◆ **Code, compile and run Java applications**

1. Get J2SDK

- u Download J2SDK from <http://java.sun.com>, the package in executable format for Windows platform, and in shell format for Unix/Linux platform

2. Install J2SDK

- u Windows: run the executable
- u Unix: run the shell

3. Set environment variables: JAVA_HOME, CLASSPATH, PATH

Ø Windows 2000/NT/XP

Start—Control panel—System—Advanced—Environment Variables

Ø Windows 9x/me:

Add these lines in autoexec.bat:

set JAVA_HOME=...

set PATH=\$JAVA_HOME\bin

set CLASSPATH=.;...

Ø Unix

csh: add these lines to \$HOME/.cshrc

setenv JAVA_HOME /usr/java...

setenv PATH \$JAVA_HOME/bin:...

setenv CLASSPATH .

bsh/ksh: add these lines to \$HOME/.profile

JAVA_HOME=/usr/java

PATH=\$JAVA_HOME/bin:\$PATH

CLASSPATH=.

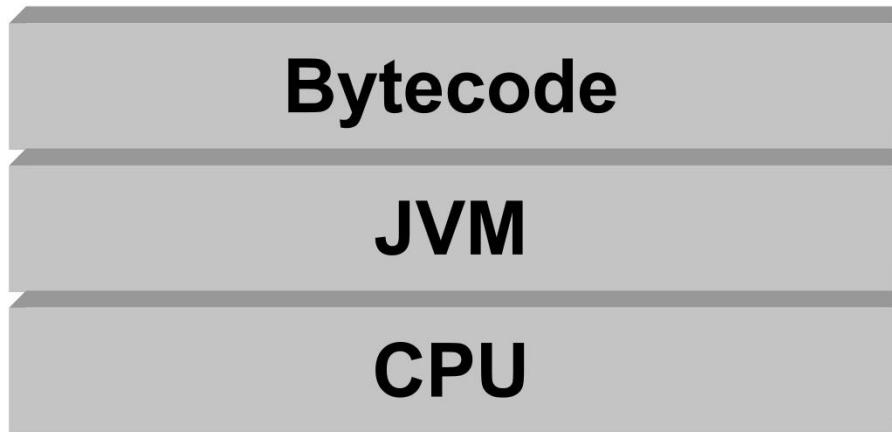
export JAVA_HOME PATH CLASSPATH



- ◆ **Programming language**
- ◆ **Development environment**
- ◆ **Application environment**
- ◆ **Deployment environment**

Why Invent Java

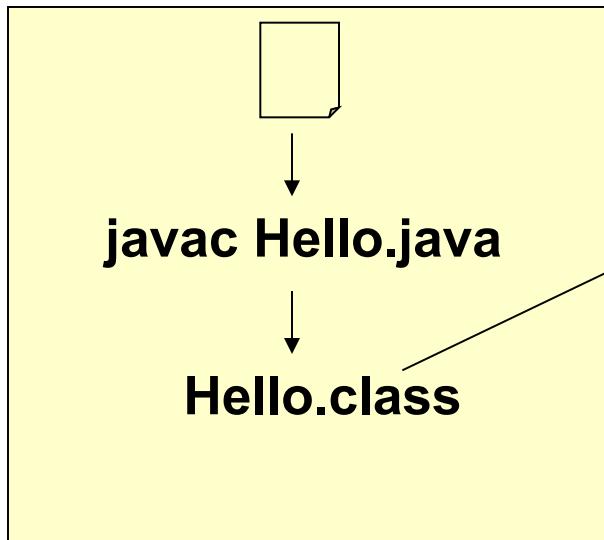
- ◆ **Provide an interpretation environment**
 - Ø Accelerate development
 - Ø Write once, run anywhere
 - Ø Multi-thread
 - Ø Dynamically support upgrading
- ◆ **Provide a easier way to program**
 - Ø More robust: no pointer, no memory management in codes, Pure object-oriented programming
- ◆ **How to implement the above**
 - Ø JVM, Garbage Collection and Code Security Verifying



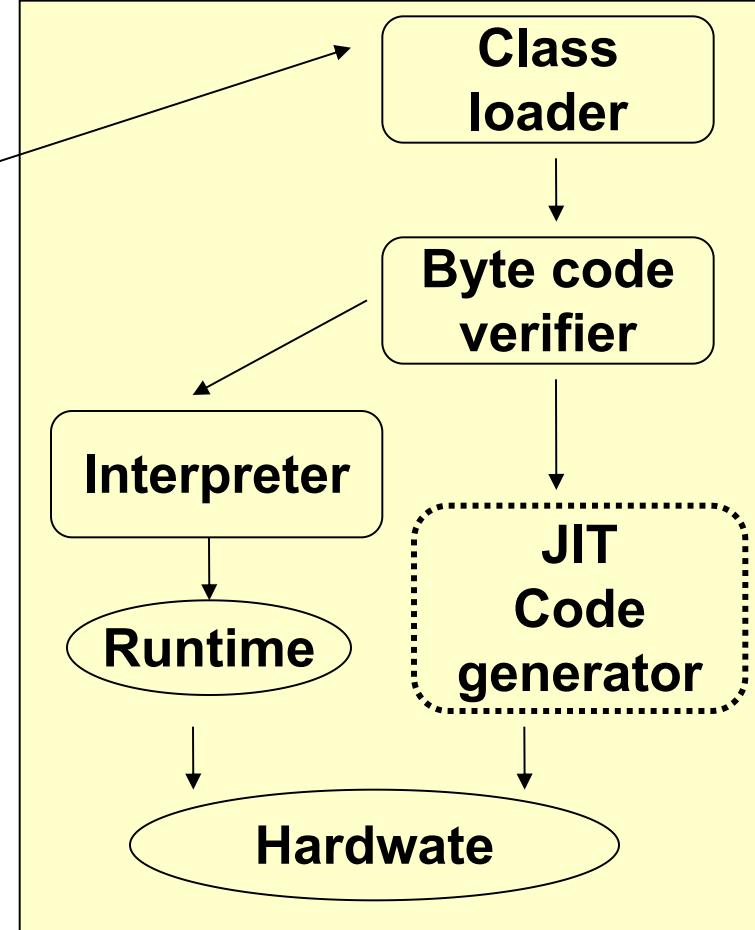
briup

- ◆ **No de-allocation of memory**
 - ∅ There is only new operator in Java
- ◆ **Java system threads automatically handle**
 - ∅ garbage collection
Suggesting Java virtual machine to expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse:
java.lang.System.gc()/java.lang.Runtime.gc()

Compile



Runtime



briup

- ◆ **Codes is compatible with JVM specifications**
- ◆ **Codes can't destroy system integrity**
- ◆ **No stack overflow and underflow**
- ◆ **Argument types are correct**
- ◆ **Type conversion is correct**



```
package sample;                                //package declaration

import java.util.Date;                         //import statements

public class HelloWorld extends Object { //class declaration

    public static void main(String[] args) { //execution entry
        System.out.println(new Date()+"Hello World!");
    }

}
```



◆ **Source files**

Can include only one top-level public class/interface, the file name has the same name as the public class/interface name it includes(if existing) and ends with .java

◆ **Package**

an entity that groups classes and interfaces together, is represented as a directory/folder in file system

1. Change path to /home/briup/corejava/chap01

2. Compile JDBees.java using javac

javac chap01\HelloWorld.java

3. Run JDBees

java chap01.HelloWorld



Important Java Commands

◆ Compile java files into class files

Ø \$ javac <options> <source files>

◆ Execute a class

Ø \$ java [-options] class [args...]

◆ Debug a class

Ø \$ jdb <options> <class> <arguments>

◆ Generate java documents

Ø \$ javadoc [options] [packagenames] [sourcefiles] [@files]

◆ Generate an archive document

Ø \$ jar {ctxu}{vfm0Mi} [jar-file] [manifest-file] [-C directory] files ...



Basic Packages in JDK

- ◆ **java.lang**
- ◆ **java.awt/javax.swing/java.awt.event**
- ◆ **java.io**
- ◆ **java.net**
- ◆ **java.util**



- u Download it from <http://java.sun.com>

Summary

- ◆ In this Chapter, we have discussed:
- ◆ How to set up Java development environment
- ◆ Java's key features
- ◆ The functions of java virtual machine
- ◆ The concepts of garbage collection
- ◆ How Java platform implements code security
- ◆ What is class, package, applets and applications
- ◆ How to code,compile and run Java applications



Review Questions

- 1. (Level 1) Which environment variables need to be set?**
- 2. (Level 1) What are the advantages of Java Language? Why Java have them?**
- 3. (Level 1) Which JDK tools can be used to compile and run a java program?**
- 4. (Level 1) What are the basic packages in Java.**
- 5. (Level 1) If all three top-level elements occur in a source file, they must appear in which order:**
 - a) A. Imports, package declarations, classes**
 - b) Classes, imports, package declarations**
 - c) package declarations; order for imports and classes doesn't matter**
 - d) package declarations, imports, classes**

Review Question-----cont.

6. (Level 2) Which of the following signatures are valid for the main() method?

- a) public static void main()
- b) public static void main(String[] args)
- c) public void main(String[] args)
- d) public static int main(String[] args)

7. (Level 3) How can you force garbage collection of an object?

- a) Garbage collection can not be forced
- b) Call Systems.gc()
- c) Call System.gc(), passing in a reference to the object to be collected
- d) Call Runtime.gc()

- 1. (Level 1) Set up your java development environment.**
- 2. (Level 2) Learn how to use JDK tools and vi to edit, compile and run FirstJavaProgram.java with package in it.**
- 3. (Level 2) Compile and run HelloWorld.java which will print out a line of text with “Hello World”.**
- 4. (Level 3) Modify HelloWorld.java to add a new method which will be called from main() and print out a line of text with “Hello World”.**



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 2

Identity、Keywords & Types

Brighten Your Way And Raise You Up.

- ◆ **Comments**
- ◆ **Identifiers**
- ◆ **Keywords**
- ◆ **Primitive data types**
- ◆ **Literal values for different types**
- ◆ **Definitions of class、object、member variable and reference variable**
- ◆ **Class declaration**
- ◆ **Class variables**
- ◆ **Create an instance**
- ◆ **Default value**
- ◆ **Describe a reference variable**

- ◆ Three types of comments

//Comments on one line

/*Comments on one or more lines*/

/documenting comments*/ (Recommended)**

- ◆ Using javadoc to create documents in HTML

javadoc [options] [packagenames] [sourcefiles] [@files]



Semicolon, Block and Whitespace

- ◆ A phrase ends with “;”
- ◆ A block of code included within “{“ “}”
- ◆ Whitespace(space, tab, newline, carriage) is insignificant

```
public class Student {  
    private String name; private String gender; private int age;  
    /** Creates a new instance of Student */  
    public Student() {}  
    /**Constructor with arguments  
     *@name--the name of a student  
     *@gender--the gender of a student  
     *@age--the age of a student  
     */  
    public Student(String name, String gender, int age) {  
        this.name = name;  
        this.gender = gender;  
        this.age = age;  
    }  
    public String toString() {  
        return "Student: " + name + " gender: " + gender + " age: " + age;  
    }  
}
```

Identifiers

- ◆ Names of class, method and variable
- ◆ Begin with character, “_” or “\$”
- ◆ Case sensitive
- ◆ No length limitation

```
package ch02;

/**
 * @author Administrator
 */
public class VariableNameTest {
    /**valid variable declaration */
    int $99;  String _ab12;

    /**invalid variable declaration */
    double 9abc;

    /** Creates a new instance of VariableNameTest */
    public VariableNameTest() {
    }
}
```

Keywords in Java

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



goto const (not used in java but reserved as keywords)

◆ Strictly, “true” and “false” are not keywords, they are literal values of boolean type

Primitive Types in Java

- ◆ **boolean** Boolean literals indicating true or false
- ◆ **byte** 8-bit integer
- ◆ **char** Stores one 16-bit unicode character
- ◆ **double** 64-bit floating-point number
- ◆ **float** 32-bit floating-point number
- ◆ **int** 32-bit integer
- ◆ **long** 64-bit integer
- ◆ **short** 16-bit integer

Boolean Data Type--boolean

- ◆ The value of a boolean variable must be either *true* or *false*
 - Ø boolean isCorrect = true;
- ◆ In C/C++, zero means *false* and non-zero means *true*



◆ char

- Ø unsigned 16-bit integer
- Ø literal value must be included within ‘ ’ and ‘ ’
- Ø `char zhChar = '中' ; char enChar = 'a'; char ucChar = '\u0060';`

◆ String

- Ø A class, not a primitive type
- Ø A string does not end with ‘\0’ like in c/c++
- Ø A string includes a sequence of unicode characters in default
- Ø `String myName = "briup";`

- ◆ Three ways to represent: Octal(8), Decimal(10), and Hexadecimal(16)
- ◆ Default type is int
- ◆ long data end with 'L' or 'L'

```
public void testIntType() {  
    int ia = 0x55;          //hex  
    int ib = 011;           //oct  
    byte bb = 0x771;  
    byte ba = (byte)0x771;  
    long la = 1234567L;     //dec  
}
```

- ◆ Default data type is double
- ◆ A float data is 32-bit long
- ◆ A double data is 64-bit long
- ◆ A float data ends with ‘f’ or ‘F’
- ◆ A double data can end with ‘d’ or ‘D’

```
public void testFloatType() {  
    float fa = 123.4f;  
    float fb = 12.5E300F;  
    float fc = (float)12.5E300;  
    double da = 123D;  
    double db = 123.456D;  
    double dc = 123.45e301;  
}
```

Type	Bits	Bytes	Minimum Range	Maximum Range
byte	8	1	-2^7	$2^7 - 1$
short	16	2	-2^{15}	$2^{15} - 1$
int	32	4	-2^{31}	$2^{31} - 1$
long	64	8	-2^{63}	$2^{63} - 1$
float	32	4	not needed	not needed
double	64	8	not needed	not needed

package sample;

```
public class Assign {
    public static void main(String args []) {
        int x, y;                                // declare int variables
        float z = 3.414f;                         // declare and assign float
        double w = 3.1415;                         // declare and assign double
        boolean truth = true;                      // declare and assign boolean
        char c;                                  // declare character variable
        String str;                            // declare String
        String str1 = "bye";                     // declare and assign String variable
        c = 'A';                                // assign value to char variable
        str = "Hi out there!";                  // assign value to String variable
        x = 6;                                 // assign values to int variables
        y = 1000;
    }
}
```

- ◆ **Class names begin with uppercase character**

class Account {...}

- ◆ **Interface names begin with uppercase character**

interface AccountBase {...}

- ◆ **Method names begin with lowercase character**

String getStudentName() {...}

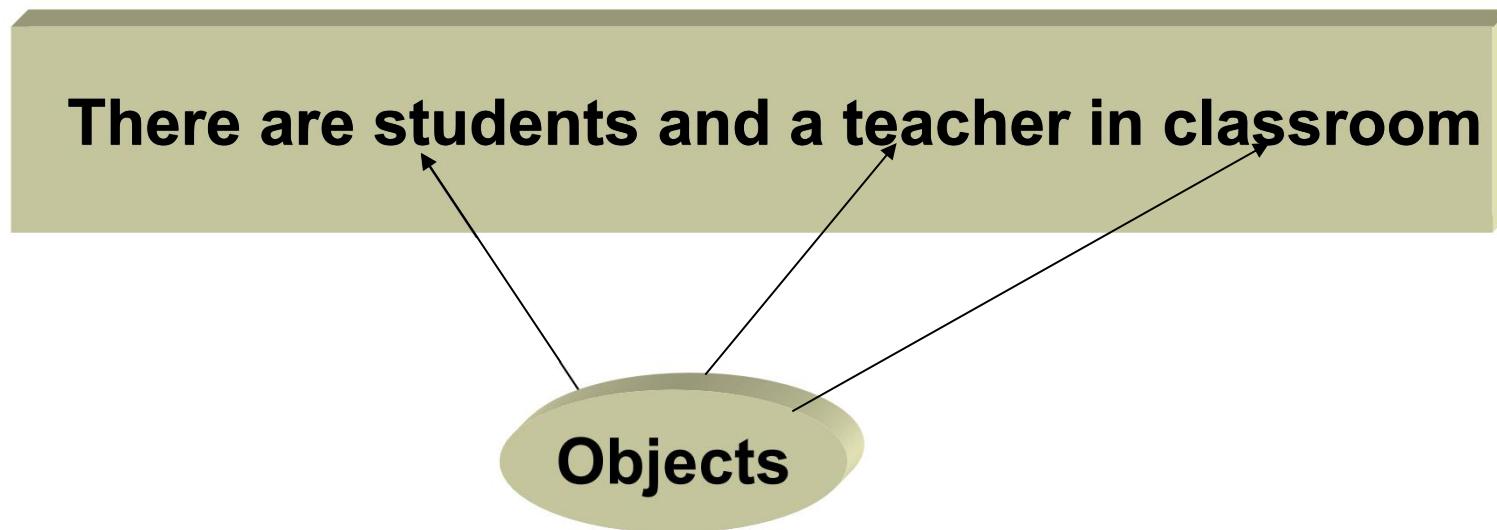
- ◆ **Variable names begin with lowercase character**

String studentName;

- ◆ **Constant variable names are in uppercase characters**

final public static int MAX_ROW = 100;

- ◆ Exist anywhere
- ◆ Nouns in most cases
- ◆ Have properties/attributes (members)
- ◆ Operations on properties/attributes (members)



The Sample Object: Teacher

briup

```
package sample;

public class Teacher {

    /**attributes of a teacher*/
    private String name;
    private int age;
    private double salary;

    /** Creates a new instance of Teacher */
    public Teacher(String name, int age, double salary) {
        this.salary = salary;
        this.age = age;
        this.name = name;
    }

    /**operations on properties */
    /** get the name of this teacher */
    public String getName() { return name; }
    /**get the salary of this teacher */
    public double getSalary() { return salary; }
    /**get the age of teacher teacher */
    public int getAge() { return age; }

    .....
}
```

- ◆ Using *new* to create an instance

```
public class Teacher {  
    .....  
    public static void main(String[] args) {  
        Teacher lz = new Teacher("Larry Zhao", 40, 10000);  
        System.out.println("Teacher: " + lz.getName());  
        System.out.println("\tAge: " + lz.getAge());  
        System.out.println("\tSalary: " + lz.getSalary());  
    }  
}
```

- ◆ **Primitive type variables evaluation**

int x = 10; int y = x;

- ◆ **Reference type variables evaluation**

Teacher lz = new Teacher("Larry Zhao", 30, 10000);

or

Teacher lz, lzClone;

lz = new Teacher("Larry Zhao", 30, 10000);

lzClone = lz;

◆ In this chapter, we have discussed:

- u Different types of comments
- u Identifier and Keyword
- u Primitive data types
- u Definition of class、object、member variable and reference variable
- u Class declaration
- u Create an instance

Review Question

- 1. (Level 1) Which type of comment is required by Javadoc?**
- 2. (Level 1) What types are primitive data types?**
- 3. (Level 1) What is the difference between variable declaration and definition?**
- 4. (Level 2) What is the relationship among class, object and reference?**
- 5. (Level 2) Choose the valid identifiers from below (choose all that apply):**
 - a) BigLongStringWithMeaninglessName**
 - b) \$int**
 - c) bytes**
 - d) finalist**

Cont.

6. (Level 2)What is the range of values for a variable of type short?

- a) A. Depends on the underlying hardware
- b) B. 0 through 2^{16} -1
- c) C. 0 through 2^{32} -1
- d) D. – 2^{15} through 2^{15} -1

7. (Level 2)What is the range of values for a variable of type byte?

- a) Depends on the underlying hardware
- b) 0 through 2^7 -1
- c) 0 through 2^{16} -1
- d) – 2^7 through 2^7 -1

Assignment

- 1. (Level 1) Learn how to use Jbuilder to write and run simple java programs..**
- 2. (Level 2)Use JDK/Vi and Jbuilder respectively, compile and run Teacher.java in you handout..**
- 3. (Level 2) Add a new method to increase the teacher's salary with 5000,then print out the new salary. In addition, add comments to your codes.**





杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 3

Expression & Flow Control

Brighten Your Way And Raise You Up.

Goals

- ◆ **Instance variables and local variables**
- ◆ **Instance variables instantiation**
- ◆ **Locate and correct errors in compiling process**
- ◆ **Operators in Java**
- ◆ **Primitive types evaluation**
- ◆ **Boolean expressions**
- ◆ **Using *if, switch, for, while, do, break* and *continue* to control flow**

- ◆ Defined in methods
- ◆ Must be initiated before being used

```
package sample;

public class LocalVariableTest {
    public void correctLocalV() {
        int x; //Just declare
        int y = 10; //Declare and initiate
        x = 20; //Evaluate
        int z = x + y; //Declare and initiate
        String str = "Hello"; //Declare and initiate
        String str1 = new String("World"); //Declare and initiate
        char ch; //Just declare
        ch = 'c'; //Evaluate
    }
    public void incorrectLocalV() {
        int x, z; //Declare only
        int y = 10;
        if( y > 20 ) z = x + y; //x isn't be initiated
        String str; //Declare only
        String str1 = "Hello" + new String(" World") + str; //str isn't be initiated
    }
}
```

- ◆ Defined in a class
- ◆ Initiated automatically with default values

byte short int long float double char boolean All reference types

0 0 0 0L 0.0f 0.0d '\u0000' false null

```
package sample;
public class InstanceVaravariableTest {
    //primitive type variables
    private int itest;
    private byte btest;
    private short stest;
    private long ltest;
    private float ftest;
    private double dtest;
    private boolean bltest;
    private char ctest;
    //reference variable
    String strTest;
    public static void main(String[] args) {
        String t = "\t";
        InstanceVaravariableTest ivTest = new InstanceVaravariableTest();
        System.out.println
            ("byte"+ t + "short" + t + "char" + t + "int" + "long" + t + "float" + t + "double" + t
             + "boolean" + t + "String");
        System.out.println
            (ivTest.btest + "\t" + ivTest.stest + "\t" + ivTest.ctest + "\t" + ivTest.itest +
             "\t" + ivTest.ftest + "\t" + ivTest.dtest + "\t" + ivTest.bltest + "\t" + ivTest.strTest);
    }
}
```

◆ Assignment operators

=	*=	/=	\$=
+=	-=	<<=	>>=
>>>=	&=	^=	=

◆ Comparison operators

>	>=	<	<=	instanceof
---	----	---	----	------------

◆ Equality operators

==	!=		
----	----	--	--

◆ Arithmetic operators

+	-	*	/	\$
---	---	---	---	----

- ◆ Shift operators

>>	<<	>>>	
----	----	-----	--

- ◆ Bitwise operators

&		^	~
---	--	---	---

- ◆ Logic operators

&&			
----	--	--	--

- ◆ Conditional operators

?	:		
---	---	--	--

- ◆ Used on primitives and instances to convert the current type
- ◆ Implicit and explicit cast
- ◆ Implicit cast is legal at runtime, from subclass type to superclass type
- ◆ Explicit cast, Narrowing conversion



```
package sample;
public class CastingTest {
    public void implicitCasting() {
        byte a = 0x60;
        int ia = a;
        char b = 'a';
        int c = b;
        long d = c;
        long e = 1000000000L;
        float f = e;
        double g = f;
        String s = "hello";
        Object o = s;
    }

    public void explicitCasting() {
        long l = 1000000L;
        int i = l; //int
        double d = 12345.678;
        float f = d; //float
        Object o = new String("Hello");
        String str = o; //String
    }
}
```

1. if (boolean experession) {
 code block
}

2. if(boolean expression) {
 code block
} else {
 code block
}

3. if(boolean expression) {
 code block
}
else if(boolean expression) {
 code block
}
else {
 code block
}



- ◆ Make sure the variable for switch is one kind type of byte, short, char, and int
- ◆ Each case statement should be followed by a break;
- ◆ The default value can be located at the end, middle, or top
- ◆ The switch—case code should like the following

```
char temp = 'c';
switch(temp) {
    case 'a': {
        System.out.println("A");
        break;
    }
    case 'b': {
        System.out.println("B");
        break;
    }
    case 'c':
        System.out.println("C");
        break;
    default:
        System.out.println("default");
}
```

◆ for loop

```
for (init_expr; boolean testexpr; alter_expr) {  
    statement or block  
}
```

◆ while loop

```
while (boolean testexpr) {  
    statement or block  
}
```

◆ do loop

```
do {  
    statement or block  
} while (boolean testexpr);
```

- ◆ **break [label]:** terminate current/specified loop
- ◆ **continue [label]:** start to the next round loop
- ◆ **label:**

```
loop: while (true) {  
    for (int i=0; i < 100; i++) {  
        switch (c = System.in.read()) {  
            case -1:  
            case `\\n` :  
                // jumps out of while-loop  
                // to line #12  
                break loop;  
                ....  
        }  
    } // end for  
} // end while
```

```
test: for (...) {  
    ....  
    while (...) {  
        if (j > 10) {  
            // jumps to the increment  
            // portion of for-loop at  
            // line #13  
            continue test;  
        }  
    } // end while  
} // end for
```

- ◆ In this Chapter, we have discussed:
- ◆ Instance variables and local variables
- ◆ Instance variables instantiation
- ◆ Operators in Java
- ◆ Primitive types evaluation
- ◆ Boolean expressions
- ◆ Using if, switch, for, while, do, break and continue to control flow

1. **(Level 1) What is the difference between instance variables and local variables?**
2. **(Level 1) What is the difference between && and &?**
3. **(Level 2) What is the difference between switch/case and if/else?**
4. **(Level 2) What is the difference between while and do loop?**
5. **(Level 2) After execution of the following code fragment, what are the values of the variables x, a, and b?**
 1. *int x,a=6,b=7;*
 2. *x=a++ + b++;*
 - a) *x=15,a=7,b=8;*
 - b) *x=15,a=6,b=7;*
 - c) *x=13,a=7,b=8;*
 - d) *x=13,a=6,b=7;*

6. (Level 2) Which of the following expressions are legal?(choose all that apply)

- a) int x=6; x=!x;
- b) int x=6; if(!(x>3)) {}
- c) int x=6; x=~x

7. (Level 3) What results after compile and run the following code?

```
1. public class Conditional {  
2.     public static void main(String args[]){  
3.         int x=4;  
4.         System.out.println("value is: "+((x>4)?99.9:9));  
5.     }  
6. }
```

- a) value is 99.9
- b) value is 9
- c) value is 9.0
- d) A compiler error at line 5

- 1. (Level 1) Using condition flow control, write a program called IfTest to test whether a number is positive, negative or 0.**

- 2. (Level 2)Using while/do/for loop flow control respectively, write a program called LoopTest to multiply 1 to 10 and print out the result.**



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 4

Array

Brighten Your Way And Raise You Up.

- ◆ **Declare and create a primitive array or a class array**
- ◆ **Why arrays should be initiated before being used**
- ◆ **Define and initiate arrays**
- ◆ **Find out how many elements in an array**
- ◆ **Create an array of array**
- ◆ **Varargs**

- ◆ A collection of the same type data
- ◆ An array is a object
 - ∅ Members: length, elements of the array
- ◆ Declaring an array does not create a object
- ◆ Arrays can be declared in the following ways
 - ∅ int[] iArray or int iArray[]
 - ∅ Teacher[] tArray or Teacher tArray[]

◆ Create a primitive array

```
int[] iArray = new int[2];
```

◆ Create a class array

```
Teacher[] teacher = new Teacher[100];
```

◆ The elements in an array are initiated as default:

- ∅ 0 for numeric type, false for boolean type and null for reference type

◆ **Declare, create and initiate separately**

```
int[] iArray;  
iArray = new int[2];  
iArray[0] = 0;  
iArray[1] = 1;
```

◆ **Declare, create and initiate in the same time**

```
int[] iArray = {0, 1};  
Student sArray[] = new Student[] {  
    new Student("Larry", "Male", 20),  
    new Student()  
};  
Student[] stArray = { new Student(), new Student() } ;
```

◆ Valid

```
int[][] ia1 = new int[2][3];
int[][] ia2 = new int[2][];
ia2[0] = new int[2], ia2[1] = new int[3];
Teacher[][] ta;
ta = new Teacher[2][];
ta[0] = new Teacher[3];
ta[1] = new Teacher[4];
```

◆ Invalid

```
int[][] ia1 = new int[][3];
Teacher[][] ta = new Teacher[][5];
```

- ◆ The index of an array begins from 0
- ◆ You can get the number of elements in an array from the attribute *length*

```
int[] iArray = new int[5];
int length = iArray.length; //length = 5;
Teacher[][] tArray = new Teacher[4][6];
length = tArray.length; //length = 4;
length = tArray[0].length; //length = 6;
```

- ◆ The size of an array can't be adjusted
- ◆ Using static method System.arraycopy()

```
// original array
```

```
int myArray[] = { 1, 2, 3, 4, 5, 6 };
```

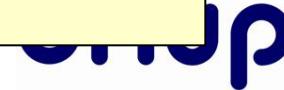
```
// new larger array
```

```
int hold[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
```

```
// copy all of the myArray array to the hold
```

```
// array, starting with the 0th index
```

```
System.arraycopy(myArray, 0, hold, 0, myArray.length);
```



u **Problem: (in pre-J2SE 5.0)**

- Ø To have a method that takes a variable number of parameters
- Ø Can be done with an array, but caller has to create it first

u **Solution: let the compiler do it**

- Ø `public static String format(String fmt, Object... args);`
- Ø Java now supports `printf(...)`



Defining a varargs method

u **Last parameter must be an array**

u **Replace [] with ...**

Old:

```
public static int sum(int[] a){  
    int r = 0;  
    for(int i = 0; i < a.length; i++)  
        r += a[i];  
    return r;  
}
```

New:

```
public static int sum(int... a){  
    int r = 0;  
    for(int i = 0; i < a.length; i++)  
        r += a[i];  
    return r;  
}
```

call:

```
sum(new int[]{1,2,3,4});
```

```
sum(1, 2, 3, 4)
```



Varargs Examples

- u APIs have been modified so that methods accept variable-length argument lists where appropriate
 - Ø `Class.getMethod(String name, Class...pTypes)`
 - Ø `Method.invoke(Object obj, Object... args)`
 - Ø `Constructor.newInstance(Object... args)`
 - Ø `Proxy.getProxyClass(ClassLoader loader, Class<?>...intfs)`
 - Ø `MessageFormat.format(String pattern, Object... args)`
- u New APIs do this too
 - Ø `System.out.printf("$d + $d = $d\n", a, b, a+b)`



- In this Chapter, we have discussed:**
- How to declare and create a primitive array or a class array**
- Why to declare and create a primitive array or a class array**
- Why arrays should be initiated before being used**
- How to define and initiate arrays**
- How to find out how many elements in an array**
- How to create an array of array**

1. (Level 1) How to define and initialize an array?
2. (Level 1) How to find out how many element in an array?
3. (Level 1) What is the first index for array?
4. (Level 3) What results from the following fragment of code?

```
1. int x=1;  
2. String[] names={"Fred","Jim","Sheila"};  
3. names[--x] +=".";  
4. for (int i=0; i<names.length;i++){  
    System.out.println(names[i]);  
6. }
```

- a) The output includes Fred. With a trailing period.
- b) The output includes Jim. With a trailing period.
- c) The output includes Sheila. With a trailing period.
- d) None of the outputs show a trailing period.

5. **(Level 2) Consider the following line of code:**

`int[] x=new int[25];`

After execution, which statements are true(Choose all that apply)

- a) x[24] is 0.**
- b) x[24] is undefined.**
- c) x[25] is 0.**
- d) x[0] is null.**
- e) x.length is 25.**

- 1. (Level 1) Write a class called ArrayMultiply. Use an array to hold 1 to 10, then multiply the ten numbers and print out the result.**
- 2. (Level 2) Write a class called ArrayCopy. Initially array A holds 1 to 10 and array B holds 10 to 1. Then copy the first 5 elements from array A to array B. Finally print out elements in array B.**



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 5

Objects & Classes

Brighten Your Way And Raise You Up.

- ◆ **Encapsulation, inheritance, polymorphism**
- ◆ ***private* and *public* modifiers**
- ◆ **Objects creation and initiation**
- ◆ **Methods invocation**
- ◆ **Constructor, overriding and overloading**
- ◆ **Keyword *this***
- ◆ **In Java programming, understand the meanings of package statement, import statement, class/member methods and fields, constructor, overloading/overriding, parent class constructor**

- ◆ **Encapsulation**
- ◆ **Inheritance**
- ◆ **Polymorphism**

briup

Abstract Data Type

- ◆ In C language, represented as struct

```
struct Student {  
    char name[32];  
    char gender[8];  
    int age;  
    /* pointers of function */  
};
```

- ◆ In Java, represented as class

```
public class Student {  
    public String name;  
    public String gender;  
    public int age;  
    //operations  
    ....  
}
```

- ◆ **Classes—abstract data type/metadata**

person

course

- ◆ **Objects—instances of classes**

“Larry Zhao” is an instance of person

“Core Java” is an instance of course

- ◆ **In Java, classes declaration and implementation at the same time**

- ◆ **In C++, classes declaration and implementation can be separated**

- ◆ <modifiers> <return_type> <name>([<argument_list>])

[throws <exception>] {<block>}

public void setName(String name) throws IllegalNameException {...}

public String getName() {...}

- ◆ return_type must be defined as void when there is no return value
- ◆ The constructor does not have return_type

Parameters Passing

```
package sample;

public class ParameterTest {
    static void increment(int i) { ++i; }
    static void changeName(Student s) {
        s.setName("Larry");
    }
    static void changeStudent(Student s) {
        s = new Student("Mary", "female", 20);
    }

    public static void main(String[] args) {
        int i = 2;
        System.out.println("before increment() " + i);
        increment(i);
        System.out.println("after increment() " + i);

        Student s = new Student("zhao", "male", 18);
        System.out.println("before changeName() " +
s);
        changeName(s);
        System.out.println("after changeName() " + s);

        s = new Student("zhao", "male", 18);
        System.out.println("before changeName() " +
s);
        changeStudent(s);
        System.out.println("after changeName() " + s);
    }
}
```

- ◆ For primitives, parameters passed by value
- ◆ For class types, parameters passed by references
- ◆ Only the content of a reference can be changed

- ◆ **this** point to the current class or object

```
public class Student {
```

```
    private String name;      private String gender;  
    private int age;
```

```
.....
```

```
    public void setName(String name) { this.name = name; }  
    public String getName() { return name; }
```

```
    public void setGender(String gender) { this.gender = gender; }  
    public String getGender() { return gender; }
```

```
    public void setAge(int age) { this.age = age; }  
    public int getAge() { return age; }
```

```
.....
```

```
}
```

```
public class Student {  
    private String name;      private String gender;  
    private int age;  
    /** Creates a new instance of Student */  
    public Student() {}  
  
    ...  
}  
  
package sample;  
public class DataHiddenTest {  
    public static void main(String[] args) {  
        Student s = new Student();  
        s.name = "Larry"; //illegal  
        s.gender = "male"; //illegal  
        s.age = 18; //illegal  
    }  
}
```

- ◆ **Making the detail of implementation invisible**
- ◆ **Uniform interface to all users**
- ◆ **Enhancing maintainability**

Method Overloading

- ◆ Same method name
- ◆ Arguments must be different
- ◆ Return type maybe different

```
package sample;  
import java.net.MalformedURLException;  
import java.net.URL;  
  
public class OverloadingTest {  
    public void print(int i) {}  
    public void print(float f) {}  
    public String print() { return "Hello"; }  
    protected URL print(String s) throws  
        MalformedURLException { return new URL(s); }  
}
```

Briup

- 1. Allocating memory space to an object, and initiating the fields of the object to 0/null/false:**
 - *new Student(), new JButton()*
- 2. Initiating the fields whose values set**
- 3. Calling the constructor**

```
package sample;

public class ObjectIniProcssTest {

    private int status = 0;
    private boolean processing = true;
    ....
}
```



- ◆ Having the same name as the class
- ◆ No return type

```
public class Xyz {  
    // member variables go there
```

```
public Xyz() {  
    // set up the object.  
}
```

```
public Xyz(int x) {  
    // set up the object with a parameter  
}
```



Constructor Overloading

- ◆ Call to this(...) must be the first statement in constructor

```
package sample;
```

```
public class Employee {  
    private String name;  
    private int salary;  
  
    public Employee(String n, int s) {  
        name = n;  
        salary = s;  
    }  
  
    public Employee(String n) {  
        this(n, 0);  
    }  
  
    public Employee() {  
        //int a = 0; //Illegal  
        this( " Unknown " );  
    }  
}
```

- ◆ Every class has constructor(s)
- ◆ Java provides a default constructor with no arguments and blank body when you do not define one

Subclass

- ◆ “Is a” relationship
- ◆ The style of declaration

- Ø Java--extends

```
public class Manager extends Employee {...}
```

- Ø C++-- :

```
class Manager : public Employee {...}
```

- ◆ Single inheritance

- ◆ Constructor can't be inherited
- ◆ Methods and fields can be inherited
- ◆ A subclass's constructor implicitly calls the default constructor of its superclass
- ◆ A subclass's constructor must explicitly call the constructor of its superclass when no default constructor exists : super(...) (must be the first statement)

See InheritanceTest.java



- ◆ The reference of its superclass
- ◆ *super* is used to call superclass's variables

See InheritanceTest.java



```
package sample;
public class InheritanceTest {
    public static void main(String[] args) {
        new SubA(1);
        SubA sa = new SubA();
        sa.print();
        new SubB();
    }
}
```



InheritanceTest.java(Cont.)

```
class SuperA {  
    private int j = 10;  
    public SuperA(int i) {System.out.println("constructor SuperA(int)"); }  
    public SuperA() { System.out.println("constructor SuperA()"); }  
    void print() {System.out.println("print() in SuperA"); }  
}  
  
class SubA extends SuperA {  
    private int k = 2;  
    public SubA(int i) { super(i); System.out.println("constructor  
        SubA(int)"); }  
    public SubA() {  
        //隐含调用super()-->SuperA()  
        System.out.println("constructor SubA()");  
    }  
    void print() { super.print(); System.out.println("print() in SubA"); }  
}
```

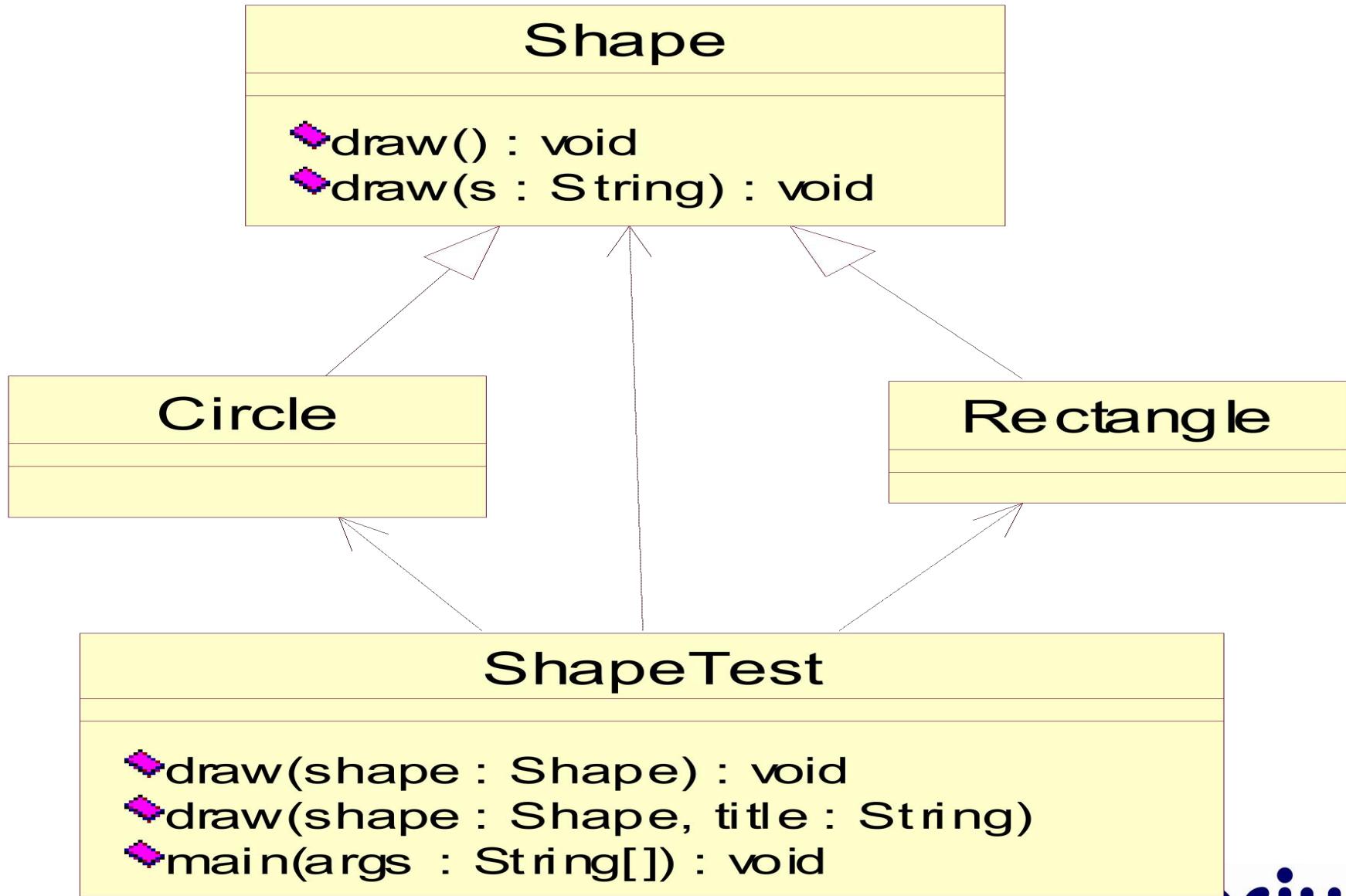
```
class SuperB {  
    public SuperB(String s) {  
        System.out.println("Constructor superB(String)");  
    }  
}  
  
class SubB extends SuperB {  
    public SubB() {  
        super("Hello");  
        System.out.println("Constructor subB()");  
    }  
}
```



- ◆ Having different types
- ◆ An object have one type
- ◆ A variable can have many types
- ◆ It is a runtime issue because of inheritance, not the same as overloading, a compiling time issue

See Shape sample





- ◆ Boolean operator
- ◆ Type identification

package sample;

```
public class InstanceofTest {  
    public static void main(String[] args) {  
        Employee unknown = new Manager();  
        Employee zhao = new Employee("zhao");  
        System.out.println("unknown is instanceof manager: " +  
                           (unknown instanceof Manager));  
        System.out.println("unknown is instanceof employee: " +  
                           (unknown instanceof Employee));  
        System.out.println("zhao is instanceof manager: " +  
                           (zhao instanceof Manager));  
        System.out.println("zhao is instanceof employee: " +  
                           (zhao instanceof Employee));  
    }  
}
```

- ◆ Using *instanceof* to identify the type
- ◆ The subtype is implicitly widened to the supertype
- ◆ The supertype must be explicitly narrowed to subtype



- ◆ The same name, argument list, and return type
- ◆ The visibility can not be narrowed
- ◆ Exceptions thrown can not be widened
- ◆ The behavior of overridden is different from that in C++

See OverridenTest



```
package sample;

import java.io.IOException;

public class OverriddenTest {
    public static void main(String[] args) {
        SuperC c = new SubC();
        c.methodA(0);
        c.methodB();
        c.methodC();
        c.methodD("hello", 3);
        c.methodE("aaa", 2);
    }
}
```



```
class SuperC {  
    public void methodA(int i) { System.out.println("methodA(int) in  
    SuperC"); }  
  
    protected void methodB() { System.out.println("methodB() in  
    SuperC"); }  
  
    void methodC() { System.out.println("methodC() in SuperC"); }  
  
    public void methodD(String s, int i) throws IOException {  
        System.out.println("methodD(String, int) in SuperC");  
    }  
  
    public int methodE(String s, int i) {  
        System.out.println("methodE(String, int) in SuperC");  
        return 0;  
    }  
}
```

OverriddenTest.java(Cont.)

```
class SubC extends SuperC {  
    public void methodA(int i) { System.out.println("methodA(int) in  
    SubC"); }  
  
    public void methodB() { System.out.println("methodB() in SubC"); }  
  
/**The followings are invalid overriding */  
    private void methodC() { System.out.println("methodC() in SubC"); }  
    public void methodD(String s, int i) throws Exception {  
        System.out.println("methodD(String, int) in SubC");  
    }  
  
    public double methodE(String s, int i) {  
        System.out.println("methodE(String, int) in SubC");  
        return 0;  
    }  
}
```

- ◆ **Implicitly invoke super() in default**
- ◆ **If constructors with arguments defined in parent class, you must explicitly invoke super(...) at the first line of subclass constructors**

- ◆ OOP basic concepts :Encapsulation, polymorphism and inheritance
- ◆ Objects creation and initiation
- ◆ Methods invocation on objects
- ◆ Constructor , overriding and overloading
- ◆ Keyword this



Review Questions

1. (Level 1) What are the three basic concepts for object oriented programming?
2. (Level 1) What are the difference between method overloading and overriding?
3. (Level 1) Does Java support multiple inheritance?
4. (Level 2) Consider this class:
 1. public class Test1{
 2. public float aMethod (float a,float b) {}
 3. }

Then Which of the following methods would be legal if added before line 3?

- a) public int aMethod (int a,int b){}
- b) public float aMethod (float a,float b) throws Exception {}
- c) public float aMethod(float a,float b ,int c) throws Exception{}
- d) public float aMethod (float c,float d) {}
- e) private float aMethod(int a,int b,int c){}

Review Questions-Cont

5. (Level 2) Consider the following class definition:

1. public class Test extends Base{}
2. public Test(int j){...}
3. public Test(int j, int k){super(j , k)}
4. }

5. 1) which of the following are legitimate calls to construct instances of the Test class ?

- A.Test t = new Test(); B.Test t = new Test(1,2,3);
- C.Test t = new Test(1,2); D.Test t = new Test(1,2,3);
- E.Test t = (new Base()).new Test(1);

 2) Which of the following forms of constructor must exist explicitly in the definition of the Base class?

- A.Base(){} B.Base(int j){}
- C.Base(int j, int k) D.Base(int j,int k, int l){}





杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Advance Java Programming

Brighten Your Way And Raise You Up.



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 1

Advance Language Features Part 1

Brighten Your Way And Raise You Up.

Goals

- ◆ **Static variables, methods and initialization blocks**
- ◆ **Static import**
- ◆ **Final classes, methods and variables**
- ◆ **Access control**
- ◆ **Abstract classes and methods**
- ◆ **Interface**
- ◆ **Inner classes**
- ◆ **Difference between “==“ and equals()**

Static Class Variables

- ◆ Shared among all class instances
- ◆ Can be accessed outside of the class
- ◆ Like global variables in some sense
- ◆ The way to access static class variables:
classname.static variable

```
package sample;

public class Count {
    private int serialNumber;
    private static int counter = 0;
    protected static int i = 1;
    public static String name = "counter";
    static String str = "clear";

    public Count() {
        counter++;
        serialNumber = counter;
    }

    public static void main(String[] args) {
        new Count();
        System.out.println("counter " + counter);
        new Count();
        System.out.println("counter " + Count.counter);
    }
}
```

Static Methods

- ◆ Can be called without an instance
- ◆ Can't access non-static variables
- ◆ Can't be overridden by not-static methods

```
package sample;

public class StaticTest {
    int x;
    public static void main(String[] args){
        System.out.println(Algorithm.add(2 , 3));
        //x = Algorithm.add(4,5); //illegal
    }
}

class Algorithm {
    public static int add(int a, int b) {
        return a + b;
    }
}
```

Static Initiation Block

- ◆ Called only once

```
package sample;
```

```
public class StaticInitTest {
```

```
    public static void main(String[] args) {
```

```
        System.out.println("Main code i = " + StaticInitDemo.i);
```

```
        System.out.println();
```

```
        System.out.println("Main code i = " + StaticInitDemo.i);
```

```
        new StaticInitDemo();
```

```
}
```

```
class StaticInitDemo {
```

```
    static int i = 5;
```

```
    static {
```

```
        System.out.println("Static code i= "+ i++ );
```

```
}
```

Static Imports

u Problem: (pre-J2SE 5.0)

- ∅ Have to fully qualify every static references from external classes or interfaces

u Solution: New import syntax

- ∅ `import static Typename.Identifier;`
- ∅ `import static Typename.*`
- ∅ Also works for static methods and enums

eg `Math.sin(x)` becomes `sin(x)`



Example for Static Imports:StaticImports.java

```
package misc;

import static java.lang.System.*;
import static java.lang.Math.*;

public class StaticImports{
    public static void main(String... args){
        out.println("Static imports test...");
        //the same as System.out.println()
        out.println("sin(PI/2) = " + sin(PI/2));
        // the same as Math.sin(Math.PI/2)
    }
}
```



Keyword final

- ◆ A final class can't be extended
- ◆ A final method can't be overridden
- ◆ A final variable can't be changed

```
package sample;
public final class FinalTest {
    final int i = 10;
    /** Creates a new instance of FinalTest */
    public FinalTest() {
    }
    public final int getNumber() {
        i = 20; //illegal
        return i;
    }
}

class FinalSub extends FinalTest {} //illegal

class FinalDemo {
    final int getNumber() {
        return 10;
    }
}

class FinalDemoSub extends FinalDemo {
    int getNumber() { //illegal
        return 20;
    }
}
```

- ◆ Methods without implementation must be declared as abstract methods

public abstract String getName();

- ◆ A class has abstract methods must be declared as abstract class

public abstract class Account{

public abstract String getName();

...

}

- ◆ An abstract class can not be instantiated

- ◆ Can declare a variable of abstract class type

- ◆ An interface is another format of a abstract class
- ◆ All methods in an interface are abstract methods
- ◆ A class can implement multiple interfaces, like multiple inheritance in C++
- ◆ All variables in an interface must be defined as final static variables
- ◆ An interface can extend multiple interfaces



Modifiers	Inside class	The same package	Subclass	other
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
Default (no modifier)	Yes	Yes	No	No
private	Yes	No	No	No

- ◆ Group related classes and thus reduce namespace clutter
- ◆ Defined at a scope smaller than a package
- ◆ An inner class can be defined inside another class, inside a method, and even as part of an expression
- ◆ There are four types of inner classes
 - ∅ static inner classes (also called nested classes)
 - ∅ member inner classes
 - ∅ local inner classes
 - ∅ anonymous inner classes

- ◆ The simplest form of inner class
- ◆ Can't have the same name as the enclosing class Compiled into a completely separate .class file from the outer class
- ◆ Can access only static members and methods of the enclosing class, including private static members
- ◆ Create an instance of a static inner class out of enclosing class:
`new outerclass.innerclass()`

```
package sample;  
import java.util.*;  
public class Week {  
    private int weeknr, year;  
  
    public Week(int weeknr, int year) {  
        this.weeknr = weeknr;  
        this.year = year;  
    }  
  
    public Iterator getDays() { return new DayIterator(this); }  
  
    public int getWeeknr() { return weeknr; }  
  
    public int getYear() { return year; }  
}
```



```
public static class DayIterator implements Iterator {  
    private int index = 0;  
    private Calendar cal = null;  
    DayIterator (Week aWeek) {  
        cal = new GregorianCalendar();  
        cal.clear();  
        cal.set(Calendar.YEAR, aWeek.year); // .getYear());  
        cal.set(Calendar.WEEK_OF_YEAR, aWeek.weeknr);  
        // .getWeeknr());  
    }  
    public boolean hasNext() { return index < 7; }  
    public Object next() {  
        cal.set(Calendar.DAY_OF_WEEK, ++index);  
        return cal.getTime();  
    }  
    public void remove() {}  
}
```

```
public static void main(String[] args) {  
    if (args.length < 2) {  
        System.err.println("Usage: java Week <weeknr> year>");  
        System.exit(1);  
    } else {  
        try {  
            int weeknr = Integer.parseInt(args[0]);  
            int year = Integer.parseInt(args[1]);  
            Week wk = new Week(weeknr, year);  
            for (Iterator i=wk.getDays();i.hasNext();) {  
                System.err.println(i.next());  
            }  
        } catch (NumberFormatException x)  
        { System.err.println("Illegal week or year");}  
    }  
}
```

Member Inner Classes

- ◆ Defined in an enclosing class without using the static modifier
- ◆ Like instance variables
- ◆ Can access all members of the enclosing class
- ◆ Create an instance within the enclosing class
this.new Innerclass();
- ◆ Create an instance out of the enclosing class
(new Outerclass()).new Innerclass();
- ◆ Access members of the enclosing class within inner classes
Outerclass.this.member

MemberInnerClassTest.java



MemberInnerClassTest.java

```
package sample;
public class MemberInnerClassTest {
    public static void main(String [] args) {
        OuterHull outer = new OuterHull();
        OuterHull.InnerCore inner = outer.new InnerCore();
        System.out.println("OuterHull = " + outer.ship);
        System.out.println("InnerCore = " + inner.ship);
        inner.communicate();
    }
}
class OuterHull {
    String ship = "U.S.S. Enterprising";
    class InnerCore {
        String ship = "Shuttle Craft";
        void communicate() {
            System.out.println("OuterHull = " + OuterHull.this.ship);
            System.out.println("InnerCore = " + this.ship);
        }
    }
}
```



- ◆ Defined within the scope a method, even smaller blocks within methods
- ◆ The least used form of inner class
- ◆ Like local variables, can't be declared public, protected, private and static
- ◆ Can only access final local variables

Week1.java



Week1.java

```
package sample;
import java.util.*;
public class Week1 { //the omitted part is almost the same as class Week
    public Iterator getDays() {
        class DayIterator implements Iterator {
            private int index = 0; private Calendar cal = null;
            DayIterator () {
                cal = new GregorianCalendar(); cal.clear();
                cal.set(Calendar.YEAR, year);
                cal.set(Calendar.WEEK_OF_YEAR, weeknr);
            }
            public boolean hasNext() { return index < 7; }
            public Object next() {
                cal.set(Calendar.DAY_OF_WEEK, index++);
                return cal.getTime();
            }
            public void remove() {}
            return new DayIterator();
        }
        ...
    }
}
```

- ◆ Local inner classes which don't have class names
- ◆ No key word **class**
- ◆ No key word **extends** and **implements**
- ◆ No constructors
- ◆ Implicitly extend a superclass or implement an interface

Week2.java

briup

Week2.java

```
package sample;
import java.util.*;
public class Week2 {
    public Iterator getDays() { //the omitted part is almost the same as class Week
        return new Iterator() { int index = 0; Calendar cal = null;
            public boolean hasNext() {
                if(cal == null ) { cal = new GregorianCalendar(); cal.clear();
                    cal.set(Calendar.YEAR, getYear());
                    cal.set(Calendar.WEEK_OF_YEAR, getWeeknr());
                }
                return index < 7;
            }
            public Object next() { cal.set(Calendar.DAY_OF_WEEK, index++);
                return cal.getTime();
            }
            public void remove() {}
        }
    ...
}
```

Primitive Type	Wrapper Class
boolean	Boolean
byte	Byte
short	Short
char	Char
int	Integer
long	Long
float	Float
double	Double

Problem: Before Autoboxing/Unboxing of Primitive Types

u Pre-J2SE 5.0

- ∅ Conversion between primitive types and wrapper types (and vice-versa)

Integer i = new Integer(2);

int iVal = i.intValue();

- ∅ You need manually convert a primitive type to a wrapper type before adding it to a collection

int i = 10;

List l = new LinkedList();

l.add(new Integer(i));



Solution: Autoboxing/Unboxing of Primitive Types

u Compiler does it

```
Integer i = 5;           //Autoboxing conversion
```

```
Long l = l;              //Unboxing conversion
```

```
Map<Integer, Double> m = new Map<Integer, Double>();
```

```
m.put(3, 10.2);         //Autoboxing conversion
```



- ◆ == identify if two references point to the same object
- ◆ *equals(Object o)* is overridden in class definition, and used to identify if two objects have the same type and content

```
package sample;
public class EqualTest {

    public static void main(String[] args) {
        String s1 = new String("Hello");
        String s2 = s1;
        String s3 = new String("Hello");
        System.out.println("s1 == s2:" + (s1==s2));
        System.out.println("s1 == s3:" + (s1==s3));
        System.out.println("s1.equals(s3):"
                           + s1.equals(s3));
        Account a = new Account("larry", 100.00);
        Account b = new Account("larry", 100.00);
        System.out.println("a == b:" + (a==b));
        System.out.println("a.equals(b):" + a.equals(b));
    }
}

class Account {
    private String name;
    private double balance;
    public Account(String name, double balance) {
        this.name = name;
        this.balance = balance;
    }
    public boolean equals(Object o) {
        Account a = (Account)o;
        return (name.equals(a.name) &&
               (balance == a.balance));
    }
}
```

- ◆ Return a string to represent a class
- ◆ Implicitly invoked when you want to print out an object

```
package sample;

public class EqualTest {
    public static void main(String[] args) {
        ...
        Account a = new Account("larry", 100.00);
        System.out.println(a);
    }
}

class Account {
    private String name;
    private double balance;
    public Account(String name, double balance) {
        this.name = name;
        this.balance = balance;
    }
    public boolean equals(Object o) {
        Account a = (Account)o;
        return (name.equals(a.name) &&
               (balance == a.balance));
    }
    public String toString() {
        return "The account: " + name +
               " has balance: " + balance;
    }
}
```

- ◆ In this chapter, we have discussed:
 - u Static variables, methods and initialization blocks
 - u Final classes, methods and variables
 - u Access control
 - u Abstract classes and methods
 - u Interface
 - u Inner classes
 - u Difference between “==” and equals()

Review Questions

- 1. (Level 1) What is the functionalities for initialization block?**
- 2. (Level 1) What is abstract method and abstract class?**
- 3. (Level 1) What is inside an interface?**
- 4. (Level 1) How many types of inner classes does Java have?**
- 5. (Level 2) Which of the following statements are true?**
 - a) An inner class may be declared private.**
 - b) An inner class may be declared static.**
 - c) An inner class defined in a method should always be anonymous.**
 - d) An inner class defined in a method can access all the method local variables.**
 - e) Construction of an inner class may require an instance of the outer class.**

Review Questions – Cont.

6. (Level 3) Consider the following definition:

```
1. public class Outer{  
2.     public int a = 1;  
3.     public int b = 2;  
4.     public void method(final int c) {  
5.         int d = 3;  
6.         class Inner{  
7.             private void iMethod(int e) {  
8.                 ...  
9.             } } } }
```

u Which variables can be referenced correctly at line 8?

- u A. a B. b C. c D. d E. e

Review Questions – Cont.

7. (Level 3) In the following code fragment, line 4 is executed.

1. String s1 = "XYZ";
2. String s2 = "XYZ";
3. If(s1 == s2)
4. System.out.println("Line 4);

A. True

B. False

Assignment

- 1. (Level 2) Modify the Shape classes in previous assignment, making Shape class an abstract class with an abstract draw() method.**
- 2. (Level 2) Keep modifying the Shape classes, making Shape class an interface.**
- 3. (Level 3) Using inner class, modify the above program with the comparator as the other three types of inner class**



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 2 Advance Language Features part 2

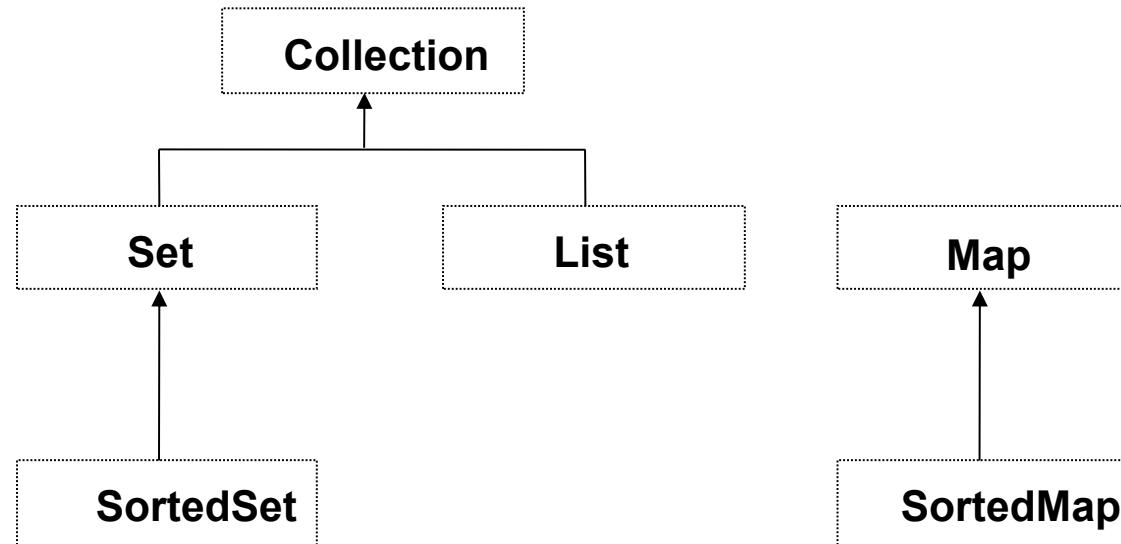
Brighten Your Way And Raise You Up.

Goals

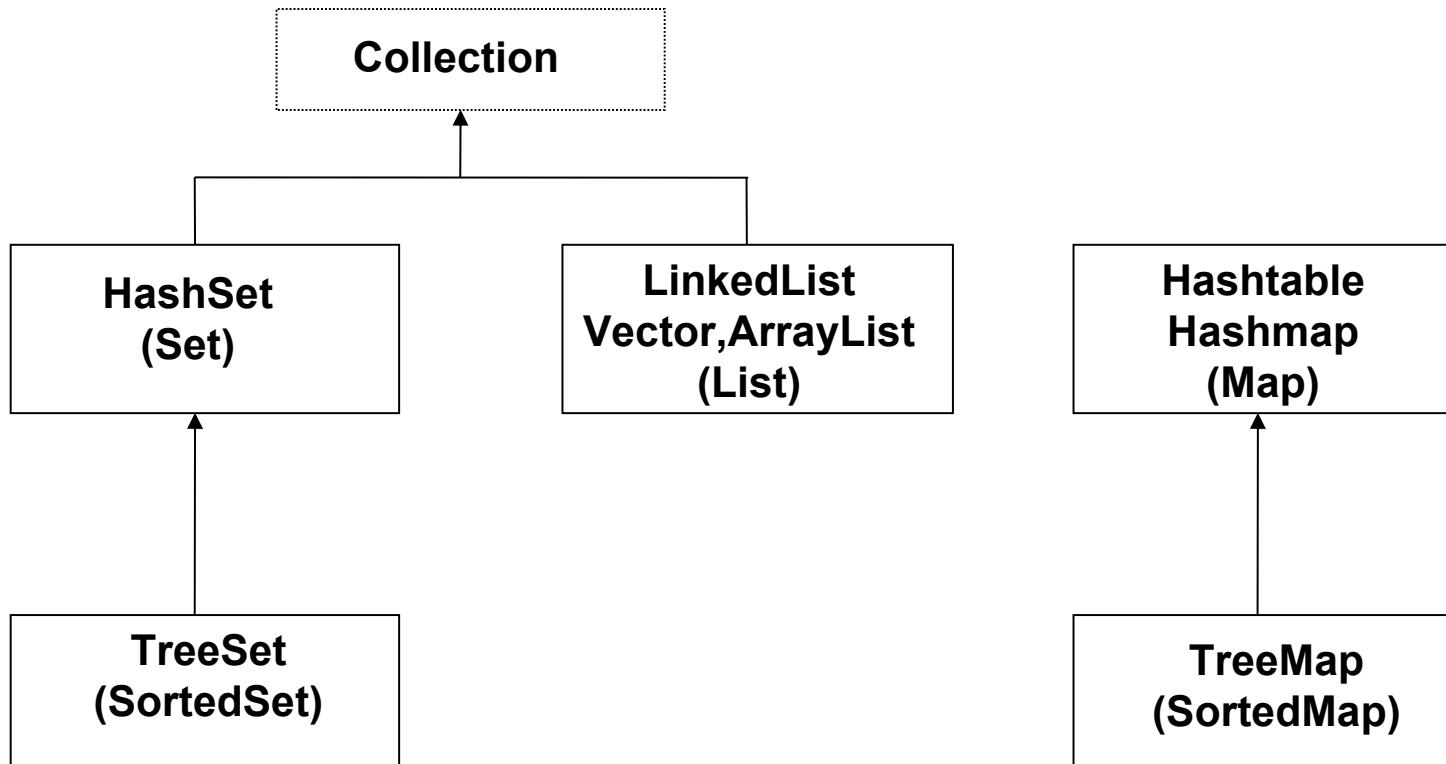
- ◆ Collection
- ◆ Enhanced for Loop
- ◆ Generics
- ◆ Enum
- ◆ Reflection

briup

- ◆ A collection is an object to store other objects and only objects
- ◆ Three elements in Java to make up collections
 - ∅ Interfaces: define the methods that each type of collection must implement
 - ∅ Implementations: actual classes such as Hashtable and Vector
 - ∅ Algorithms: methods that store, retrieve, and manipulate elements in a collection
- ◆ Package **java.util**



- u **Set:** does not allow duplicate objects to enter the collection of elements
- u **SortedSet:** similar to set except that the elements in the set are stored in ascending order
- u **List:** is ordered, maintain an order as objects are added and removed from the collection can also contain duplicate entries of objects
- u **Map:** stores objects that are identified by unique keys, and may not store duplicate keys
- u **SortedMap:** similar to Map, except the objects are stored in ascending order according to their keys



Collection Example: List

```
package sample;

import java.util.*;

public class ArrayListTest {

    public static void main(String[] args) {

        List l = new ArrayList();
        l.add(new Integer(1));
        l.add(new Integer(4));
        l.add(new Integer(3));
        l.add(new Integer(2));

        Iterator it = l.iterator();
        while(it.hasNext()){
            int i = ((Integer)it.next()).intValue();
            System.out.println("Element in list is: " + i);
        }
    }
}
```

Collection Example : Map

```
package sample;

import java.util.*;

public class MapTest {
    public MapTest() {
    }

    public static void main(String[] args) {
        String[] s1 = {
            "Wang", "Zhang", "Li"};
        String[] s2 = {
            "78", "87", "88"};
        Map hm = new HashMap();

        for (int i = 0; i < s1.length; i++) {
            hm.put(s1[i], s2[i]);
        }

        System.out.println("Wang score is: " + hm.get("Wang"));
        System.out.println("Zhang score is: " + hm.get("Zhang"));
        System.out.println("Li score is: " + hm.get("Li"));
    }
}
```

Enhanced for Loop (foreach)

u Problem: Pre-J2SE 5.0

- Ø Iterating over collections is tricky
- Ø Often, iterator only used to get an element
- Ø Iterator is error prone

u Solution: let the compiler do it

- Ø New for loop syntax

```
for(variable : collection)
```

- Ø Works for Collections and arrays



Example for Enhanced for Loop:NewLoop.java (1)

```
package misc;

import java.util.Collection;
import java.util.ArrayList;
import java.util.Iterator;

public class NewLoop{
    static public void oldPrint(Collection c){
        //old style for loop
        for(Iterator i = c.iterator(); i.hasNext();){
            Integer in = (Integer)i.next();
            System.out.println(in);
        }
    }

    //enhanced for loop
    static public void newPrint(Collection<Integer> c){
        //enhanced for loop and unboxing
        for(int i : c)    System.out.println(i);
    }
}
```



Example for Enhanced for Loop:NewLoop.java (2)

```
public static void main(String[] args){  
    Collection c = new ArrayList();  
    for(int i = 0; i < 5; i++) c.add(new Integer(i));  
    oldPrint(c);  
  
    Collection<Integer> ic = new ArrayList<Integer>();  
    for(int i = 0; i < 5; i++) ic.add(i); //autoboxing  
    newPrint(c);  
}  
}
```



What are Generics?

- u **Generics provide abstraction over types**
 - ø Classes, Interfaces and Methods can be **Parameterized by Types** (in the same way a Java type is parameterized by an instance of it)
- u **Generics make type safe code possible**
 - ø If it compiles without any errors or warnings, then it must not raise any unexpected ClassCastException during runtime
- u **Generics provide increased readability**

Definition and Usage of Generics

- u **Definition:** `LinkedList<E>` has a type parameter E that represents the type of the elements stored in the list

```
public class LinkedList<E> {  
    void add(E x);  
    Iterator<E> iterator();  
}
```

- u **Usage:** Replace `type parameter<E>` with **concrete type argument**, like `<Double>` or other types

- ∅ `LinkedList<Double>` can store only double or sub-type of Double as elements

```
LinkedList<Double> dl = new LinkedList<Double>();  
dl.add(new Double(0.1));  
Double d = dl.iterator().next();
```

What Generics are not

- u **Generics are not templates**
- u **Unlike c++, generic declarations are type checked**
- u **Generics are compiled once and for all**
- u **Generic source code are not exposed to user**
- u **No bloat required**

- Before J2SE 5.0, the following code is not type safe:

```
ArrayList al = new ArrayList();
```

```
al.add("Hello");
```

```
al.add(new Double(1.2));
```

```
al.add(new Object());
```

// ClassCastException occurs during runtime

```
String s = (String)al.get(1);
```

Generics Solve ...

u Problem: Collection element types

- Ø Compiler is unable to verify types
- Ø Assignment must have type casting
- Ø ClassCastException can occur during runtime

u Solution: Generics

- Ø Tell the compiler type of the collection
- Ø Let the compiler fill in the cast

Using Generic Classes:

- u Instantiate a generic class to create type specific object
- u In J2SE 5.0, all collection classes are rewritten to be generic classes

Using Generic Classes: Group.java

```
package generics;
```

```
import java.util.HashSet;
```

```
import java.util.Iterator;
```

```
import commons.Student;
```

```
public class Group{
```

```
    private HashSet<Student> students;
```

```
    public Group(){ students = new HashSet<Student>(); }
```

```
    public void join(Student student){ students.add(student); }
```

```
    public void leave(Student student){ students.remove(student); }
```

```
    public Iterator<Student> iterator(){ return students.iterator(); }
```

```
}
```



Using Generic Classes: Student.java (1)

```
package commons;

import java.util.Date;
import java.io.Serializable;

public class Student implements Serializable{
    private Integer id;
    private String name;
    private String sex;
    private Date birthday;

    public Student(){}
    public Student(String name, String sex, Date birthday){
        this.name = name;
        this.sex = sex;
        this.birthday = birthday;
    }
}
```



Using Generic Classes: Student.java(2)

```
public void setId(Integer id){ this.id = id; }
    public Integer getId() { return id; }

    public void setName(String name){ this.name = name; }
    public String getName(){ return name; }

    public void setSex(String sex){ this.sex = sex; }
    public String getSex() { return sex; }

    public void setBirthday(Date birthday) { this.birthday = birthday; }
    public Date getBirthday() { return birthday; }

@Override public boolean equals(Object o){ //using annotation
    if(!(o instanceof Student)) return false;

    Student s = (Student)o;
    return id.equals(s.id) && name.equals(s.name)
        && sex.equals(s.sex) && birthday.equals(birthday);
}

@Override public int hashCode(){ return (name + sex + birthday.toString()).hashCode(); }
}
```



Using Generic Classes:

- u **Generic class can have multiple type parameters**
- u **Type argument can be a custom type**

```
public class HashMap<K, V>{  
    public V put(K key, V value);  
    public V get(Object key);  
}
```



Using Generic Classes: GroupMgmt.java (1)

```
package generics;

import java.util.HashMap;
import java.util.Iterator;
import commons.Student;
import java.util.Date;
import java.util.Calendar;
import static java.util.Calendar.*; //static import

public class GroupMgmt{
    public static Date mkDate(int year, int month, int day){
        Calendar cal = Calendar.getInstance();
        //statically importing constants
        cal.set(YEAR, year);
        cal.set(MONTH, month - 1);
        cal.set(DAY_OF_MONTH, day);

        return cal.getTime();
    }
}
```



Using Generic Classes: GroupMgmt.java (2)

```
public static void main(String[] args){  
    HashMap<Integer, Group> gmap = new HashMap<Integer, Group>();  
    Group g = new Group();  
    g.join(new Student("larry1", "Male", mkDate(1970, 3, 21)));  
    gmap.put(1, g); //autoboxing  
  
    Group g0 = new Group();  
    g0.join(new Student("Larry", "Male", mkDate(1966, 2, 11)));  
    g0.join(new Student("Mary", "Female", mkDate(1974, 6, 2)));  
    gmap.put(2, g0); //autoboxing  
  
    Group group = gmap.get(1); //autoboxing  
    Iterator<Student> i = group.iterator();  
    while(i.hasNext()){  
        System.out.println(i.next().getName());  
    }  
}
```

Subtyping

u The following is correct (pre-J2SE 5.0)

Ø Object o = new Integer(3);

u The following is correct, too (pre-J2SE 5.0)

Ø Object[] oarr = new Integer[5];

u So you would expect to be able to do like the following (Ooooooops,
you can't do this !!!)

Ø ArrayList<Object> ao = new ArrayList<Integer>();

Ø This is counter-intuitive at the first glance

Subtyping

- u Why this compile error? It is because if it si allowed, `ClassCastException` can occur during runtime – this is not type-safe

```
ArrayList<Integer> al = new ArrayList<Integer>();
```

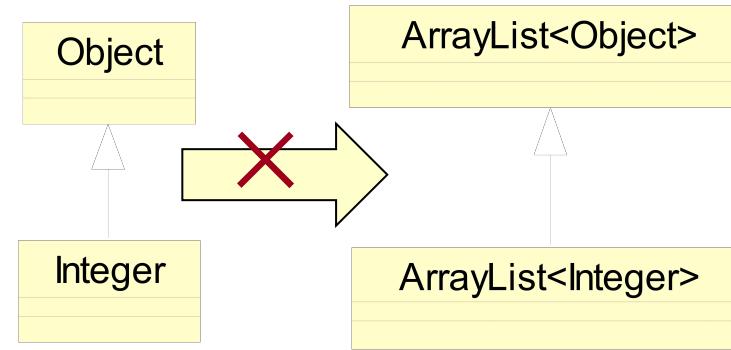
```
ArrayList<Object> ao = al; //If it is allowed at compile time
```

```
ao.add(new Object());
```

```
Integer l = al.get(0); //this would result in runtime
```

// `ClassCastException`

- u so there is no inheritance relationship between type arguments of a generic class



Subtyping

u The following code work

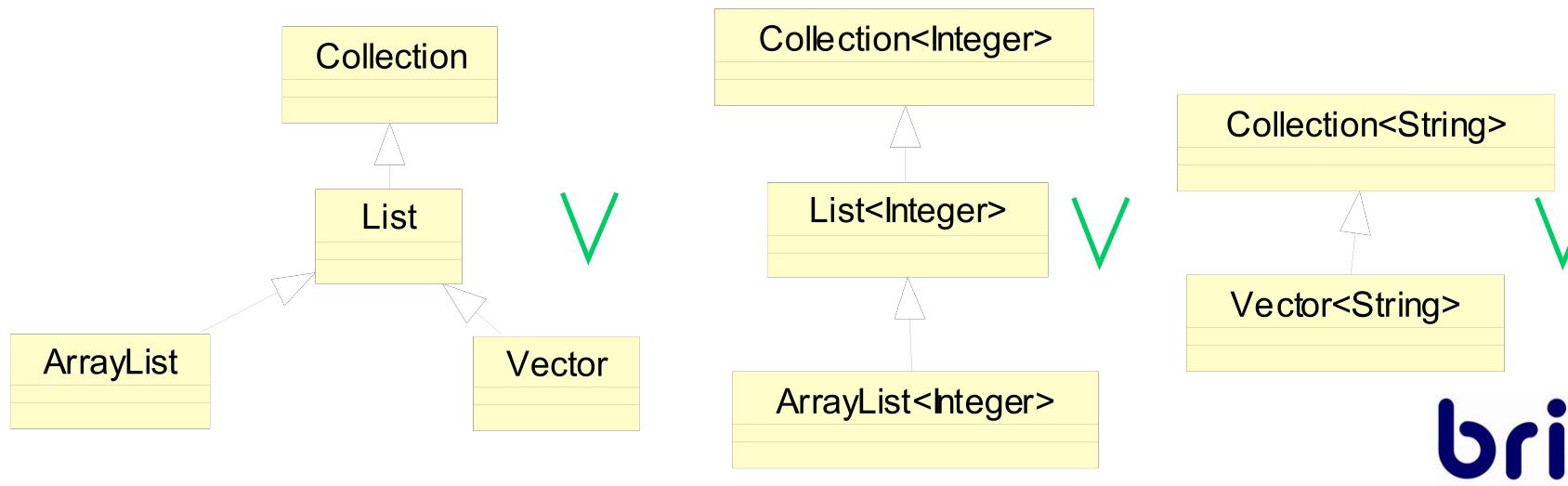
```
ArrayList<Integer> ai = new ArrayList<Integer>();
```

```
List<Integer> li = new ArrayList<Integer>();
```

```
Collection<Integer> ci = new ArrayList<Integer>();
```

```
Collection<String> cs = new Vector<String>(16);
```

u Inheritance relationship between generic classes themselves still exist



u **The following code work**

```
ArrayList<Number> an = new ArrayList<Number>();  
an.add(new Integer(4));  
an.add(new Long(10000L));  
An.add(new String("World")); //compile error
```

u **Entries in collection maintain inheritance relationship**

- u **How do you print contents of any Collection – Collection of any type?**
 - ∅ The following code does not work (see the next slide)

Problem: ErrorGenerics.java

```
package generics

import java.util.Collection;
import java.util.List;
import java.util.ArrayList;
import java.util.Vector;
import static java.lang.System.out;

public class ErrorGenerics{
    static void printCollection(Collection<Object> c){
        for(Object o : c)
            out.println(o);
    }

    public static void main(String... args){
        List<Integer> li = new ArrayList<Integer>(10);
        printCollection(li); //Compile error
        Collection<String> cs = new Vector<String>();
        printCollection(cs); //Compile error
    }
}
```



Solution: Wildcards <?>

- u **Use wildcard type argument <?>**
- u **Collection<?> means Collection of unknown type**

Solution – Wildcards <?>:WildcardsGenerics.java

```
package generics;

import java.util.Collection;
import java.util.List;
import java.util.ArrayList;
import java.util.Vector;
import static java.lang.System.out;

public class WildcardsGenerics{
    static void printCollection(Collection<?> c){
        for(Object o : c)
            out.println(o);
    }

    public static void main(String... args){
        List<Integer> li = new ArrayList<Integer>(10);
        printCollection(li);
        Collection<String> cs = new Vector<String>();
        printCollection(cs);
    }
}
```



u **New problem**

- ∅ How do you write a method to print contents of collection of only Number type?

u **Solution: Bounded wildcards**

- ∅ Use <? extends Number> type argument

Bounded Wildcards <? extends T>:BoundedGenerics.java

```
package generics;
import java.util.Collection;
import java.util.List;
import java.util.ArrayList;
import java.util.Vector;
import static java.lang.System.out;

public class BoundedGenerics{
    static void printCollection(Collection<? extends Number> c){
        for(Object o : c) out.println(o);
    }

    public static void main(String... args){
        List<Integer> li = new ArrayList<Integer>(10);
        printCollection(li);
        List<Long> ll = new ArrayList<Long>();
        printCollection(ll);
        //Collection<String> cs = new Vector<String>();
        //printCollection(cs); // You want compile error
    }
}
```



- u The element that is added must be type (or sub-type) of unknown type x, but x is unknown to the compiler, so compiler disallows adding any type except null

package generics;

import java.util.ArrayList;

public class ErrorUnknown{

public static void main(String... args){

ArrayList<?> a1 = new ArrayList<Number>(10);

a1.add(new Integer(3)); // Compile error

a1.add(new Long(100L)); // Compile error

a1.add(new Object()); // Compile error

a1.add(null);

Object o = a1.get(0);

Number n = a1.get(0); // Compile error

Integer i = a1.get(0); // Compile error

}

}

briup

- u Add type parameters in class definition
- u Typically use one capital letter for type parameter
- u Use type parameters anywhere in class that type is required

Defining Generic Classes:EntrySet.java

```
package generics;
```

```
public class EntrySet<K, V>{
```

```
    private K key;
```

```
    private V value;
```

```
    public EntrySet(K key, V value){
```

```
        this.key = key;
```

```
        this.value = value;
```

```
}
```

```
    public void setKey(K key) { this.key = key; }
```

```
    public K getKey() { return key; }
```

```
    public void setValue(V value) { this.value = value; }
```

```
    public V getValue() { return value; }
```

```
}
```



Usage of Generic the Classes:EntrySetTest.java

```
package generics;

import static java.lang.System.out;

public class EntrySetTest{
    public static void main(String... args){
        EntrySet<Integer, String> es =
            new EntrySet<Integer, String>(90, "Hello");
        out.println("key: " + es.getKey());
        out.println("value: " + es.getValue());
        es.setValue("Aloha");
        out.println("After change value: ");
        out.println("value: " + es.getValue());
    }
}
```

```
package generics;

import java.util.Comparator;
import java.util.Date;
import commons.Student;

public class StudentAgeComparator<T extends Student>
    implements Comparator<T>{
    public int compare(T o1, T o2){
        return -o1.getBirthday().compareTo(o2.getBirthday());
    }

    public boolean equals(Object o){
        return o instanceof StudentAgeComparator;
    }
}
```



```
package generics;

import java.util.Comparator;
import java.util.Date;
import commons.Graduate;
import static java.lang.System.out;
import static generics.GroupMgmt.mkDate;

public class StudentAgeComparatorTest{
    public static void main(String[] args){
        Graduate zhao = new Graduate("larry1", "Male", mkDate(1968, 3, 21));
        Graduate larry = new Graduate("Larry", "Male", mkDate(1964, 8, 10));
        StudentAgeComparator<Graduate> comp =
            new StudentAgeComparator<Graduate>();
        int result = comp.compare(zhao, larry);
        out.println(result >= 0 ? "larry1 is older than Larry" :
                    "larry1 is younger than Larry");
    }
}
```

```
package commons;

import java.util.Date;
import java.io.Serializable;

public class Graduate extends Student implements Serializable{
    private Integer teacherId;

    public Graduate(){}
    public Graduate(String name, String sex, Date birthday){
        super(name, sex, birthday);
    }

    public void setTeacherId(Integer teacherId){ this.teacherId = teacherId; }
    public Integer getTeacherId() { return teacherId; }
}
```



Raw Type

- u **Generic type instantiated with no type arguments**
- u **Pre-J2SE 5.0 classes continue to function over J2SE 5.0 JVM as raw type**

//Generic type instantiated with type argument

```
List<String> ls = new LinkedList<String>();
```

//Generic type instantiated with no type argument—this is raw //type

```
List lraw = new LinkedList();
```



Type Erasure

- u All generic type information is removed in the resulting byte-code after compilation
- u So generic type information does not exist during runtime
- u After compilation, they all share same class
 - Ø The class that represents ArrayList<String>, ArrayList<Integer> is the same class that represent ArrayList

So, the Output is true: TypeEraseClassEquals.java

```
package generics;  
import java.util.ArrayList;  
  
public class TypeEraseClassEquals{  
    public static void main(String[] args){  
        ArrayList<Integer> ai = new ArrayList<Integer>();  
        ArrayList<String> as = new ArrayList<String>();  
        boolean eq = (ai.getClass() == as.getClass());  
        System.out.println("Do ArrayList<Integer> and ArrayList<String> share the same  
class: " + eq);  
    }  
}
```



Type-safe Code

- u **The compiler guarantees that either**
 - ∅ The code it generates will be type-correct at run time, or
 - ∅ It will output a warning (using Raw type) at compile time
- u **If the code compiles without warnings and has no casts, then you will never get a ClassCastException**
 - ∅ This is “type safe” code

- u **All existing pre-J2SE 5.0 code will compile with J2SE 5.0 compiler**
 - ∅ With one exception (enum is now keyword)
 - ∅ And some obscure cases that won't arise in practice

- u For raw type, compiler does not have enough type information (for type checking), so it just generates “unchecked” or “unsafe” warning
- u If you ignore them, ClassCastException can still occur during runtime

- u **Pre-J2SE 5.0 class files run over J2SE 5.0 platform**
 - Ø Backward binary compatibility is preserved
- u **Pre-J2SE 5.0 classes and J2SE 5.0 classes should be able to call each other**
 - Ø In a simple application the pre-J2SE 5.0 classes and libraries and J2SE 5.0 classes and libraries can coexist and interoperate

- u **Painless migration. You can make your code generic without waiting for anyone else**
- u **Do not use raw type unless you have to still can have unsafe code**
- u **If you have to use raw type or have to use pre-J2SE 5.0 compiled classes or libraries, make sure the “unsafe” compile warnings are really just warning**

Type-Safe Enumerations

- u **Problem: (Pre-J2SE 5.0) Previously, if you wanted to define an enumeration you**
 - Ø **Defined a bunch of integer constants**
 - Ø **Followed one of the various “type-safe enum patterns”**
- u **Issues of using integer constants**
 - Ø **public static final int CAR_AUDI = 0;**
 - Ø **public static final int CAR_GOLF = 1;**
 - Ø **Not type safe (any integer will pass)**
 - Ø **No namespace (CAR_*)**
 - Ø **Brittleness (how do add value in-between?)**
 - Ø **Printed values uninformative (prints just int values)**



Type-Safe Enumerations

u Issues of using “type-safe enum patterns”

- ∅ Verbose
- ∅ Do not work well with switch statements

u Solution: New type of class declaration

- ∅ enum type has public, self-typed members for each enum constant
- ∅ New keyword, enum



New enum Class

- u New reserved work **enum** added to the language
- u An enum is a special kind of class
- u It can be declared anywhere an interface can
 - ∅ At the top level within a package
 - ∅ Inside a class
 - ∅ But not inside a method
- u enum classes cannot be subclassed



Example for Type-Safe enum 1:Card.java (1)

```
package misc;  
import java.util.List;  
import java.util.ArrayList;  
  
public class Card {  
    public enum Rank { DEUCE, THREE, FOUR, FIVE, SIX,  
        SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING, ACE }  
  
    public enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES }  
  
    private final Rank rank;  
    private final Suit suit;  
    private Card(Rank rank, Suit suit) {  
        this.rank = rank;  
        this.suit = suit;  
    }  
  
    public Rank rank() { return rank; }  
    public Suit suit() { return suit; }  
    public String toString() { return rank + " of " + suit; }
```



Example for Type-Safe enum 1:Card.java (2)

```
private static final List<Card> protoDeck = new ArrayList<Card>();  
  
// Initialize prototype deck  
static {  
    for (Suit suit : Suit.values())  
        for (Rank rank : Rank.values())  
            protoDeck.add(new Card(rank, suit));  
}  
  
public static ArrayList<Card> newDeck() {  
    return new ArrayList<Card>(protoDeck);  
    // Return copy of prototype deck  
}  
}
```



Example for Type-Safe enum 1:Play Card--Deal.java (1)

```
package misc;
```

```
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

public class Deal {
    public static void main(String args[]) {
        if(args.length != 2){
            System.out.println("Usage: java " + Deal.class.getName() +
                " numHands cardPerHand");
            System.exit(1);
        }
        int numHands = Integer.parseInt(args[0]);
        int cardsPerHand = Integer.parseInt(args[1]);
        List<Card> deck = Card.newDeck();
        Collections.shuffle(deck);
        for (int i=0; i < numHands; i++)
            System.out.println(deal(deck, cardsPerHand));
    }
}
```



```
public static ArrayList<Card> deal(List<Card> deck, int n) {  
    int deckSize = deck.size();  
    List<Card> handView = deck.subList(deckSize-n, deckSize);  
    ArrayList<Card> hand = new ArrayList<Card>(handView);  
    handView.clear();  
    return hand;  
}  
}
```



New classes for enums

- u All enums inherit from java.lang.Enum
- u Each enum type has a static values method that returns an array containing all of the values of the enum type in the order they are declared
 - Ø A value in the array is a static instance of enum inside the enum
- u You can add methods and fields to an enum type
- u An enum type can implement interfaces
- u Java.util.EnumSet is a set of enum values
 - Ø All from the same enum definition
- u Java.util.EnumMap is map where the keys are enum values
 - Ø All from the same enum definition



Example for Type-Safe enum 2:TrafficLight.java

```
package misc;
public enum TrafficLight implements java.io.Serializable {

    RED(30){ public TrafficLight next() { return GREEN; } },
    AMBER(10){ public TrafficLight next() { return RED; } },
    GREEN(40){ public TrafficLight next() { return AMBER; } };

    private final int duration;
    TrafficLight(int duration){ this.duration = duration; }

    public int duration() { return duration; }

    public abstract TrafficLight next();

    public static void main(String[] args){
        for(TrafficLight light : TrafficLight.values()){
            System.out.print(light);
            System.out.println( "\t" + light.duration());
        }
    }
}
```



- ◆ Determine the class of an object
- ◆ Get information about a class's modifiers, fields, methods, constructors, and superclasses
- ◆ Find out what constants and method declarations belong to an interface
- ◆ Create an instance of a class whose name is not known until runtime
- ◆ Get and set the value of an object's field, even if the field name is unknown to your program until runtime
- ◆ Invoke a method on an object, even if the method is not known until runtime
- ◆ Create a new array, whose size and component type are not known until runtime, and then modify the array's components

- ◆ **java.lang.Class**
- ◆ **java.lang.reflect.Field**
- ◆ **java.lang.reflect.Method**
- ◆ **java.lang.reflect.Array**
- ◆ **java.lang.reflect.Constructor**



```
package sample;
import java.lang.reflect.*;
public class ReflectionTest {
    public static void main(String[] args) {
        if(args.length == 0 ){
            System.out.println("Usage: java ReflectionTest class_name");
            System.exit(-1);
        }
        String name = args[0];
        try {
            Class cl = Class.forName(name); Class supercl = cl.getSuperclass();
            System.out.print("class " + name);
            if (supercl != null && !supercl.equals(Object.class))
                System.out.print(" extends " + supercl.getName());
            System.out.print("\n{\n");
            printConstructors(cl); System.out.println();
            printMethods(cl);   System.out.println();
            printFields(cl);  System.out.println("}");
        } catch(ClassNotFoundException e) {
```

```
System.out.println("Class not found.");    }
}
public static void printConstructors(Class cl)  {
Constructor[] constructors = cl.getDeclaredConstructors();
for (int i = 0; i < constructors.length; i++) {
Constructor c = constructors[i];
Class[] paramTypes = c.getParameterTypes();
String name = c.getName();
System.out.print(Modifier.toString(c.getModifiers()));
System.out.print(" " + name + "(");
for (int j = 0; j < paramTypes.length; j++) {
if (j > 0) System.out.print(", ");
System.out.print(paramTypes[j].getName());
}
System.out.println(");");
}
}
```



```
public static void printMethods(Class cl) {  
    Method[] methods = cl.getDeclaredMethods();  
    for (int i = 0; i < methods.length; i++) {  
        Method m = methods[i];  
        Class retType = m.getReturnType();  
        Class[] paramTypes = m.getParameterTypes();  
        String name = m.getName();  
        System.out.print(Modifier.toString(m.getModifiers()));  
        System.out.print(" " + retType.getName() + " " + name  
                + "(");  
        for (int j = 0; j < paramTypes.length; j++)  
        { if (j > 0) System.out.print(", ");  
            System.out.print(paramTypes[j].getName());  
        }  
        System.out.println(");");  
    }  
}
```

```
public static void printFields(Class cl) {  
    Field[] fields = cl.getDeclaredFields();  
  
    for (int i = 0; i < fields.length; i++) {  
        Field f = fields[i];  
        Class type = f.getType();  
        String name = f.getName();  
        System.out.print(Modifier.toString(f.getModifiers()));  
        System.out.println(" " + type.getName() + " " + name  
            + ":" );  
    }  
}
```



Summary

- ◆ In this chapter, we have discussed:
- ◆ Collection
- ◆ Enhanced for Loop
- ◆ Generics
- ◆ Enum
- ◆ Reflection



Review Questions

- 1. (Level 1) What are the differences between Map, Set and List?**

- 2. (Level 2) Which would be most suitable for storing data elements that must not appear in the store more than once, if searching is not a priority?**

A. Collection B. List C. Set D. Map E. Vector

Assignment

- 1. (Level 3) Using Reflection, write a class called TestReflect which has a method called dynCall() to dynamically call a method in another class.**
- 2. (Level 2) Using TreeSet and Comparator, write a class called TreeSetTest to sort 6 numbers(2,4,6,1,5,3) ascendingly and descedingly.**
- 3. (Level 3)Using inner class,write a class called TreeSetTestInner2 to modify the above program with the comparator as member inner class**
- 4. (Level 2) Using HashMap, write a class called HashMapTest to print out key/value pairs.**
- 5. (Level 3) Using Map.Entry inner class, write a class called MapEntryTest to print out key/value pairs.**



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 3

Exceptions

Brighten Your Way And Raise You Up.

- ◆ **Defining exceptions**
- ◆ **Using *try*, *catch* and *finally* statements**
- ◆ **Exception categories**
- ◆ **Exceptions handling**
- ◆ **Assertion**

- ◆ **Exceptional conditions**
- ◆ **Occurrences that alter the normal program flow**
- ◆ **When an exceptional event occurs, an exception is said to be thrown**
- ◆ **The code that is responsible for doing something about the error is called an exception handler**
- ◆ **The exception handler catches the exception**

```
1. try {  
2.   // This is the first line of the “guarded region”  
3.   // that is governed by the try keyword.  
4.   // Put code here that might cause some kind of  
exception.  
5.   // We may have many code lines here or just one.  
6. } catch(MyFirstError) {  
7.   // Put code here that handles this error.  
8.   // This is the next line of the exception handler.  
9.   // This is the last line of the exception handler.  
10. } catch(MySecondError)  
11.   // Put code here that handles this error.  
12. }  
13. // Some other unguarded code begins here
```

finally statement

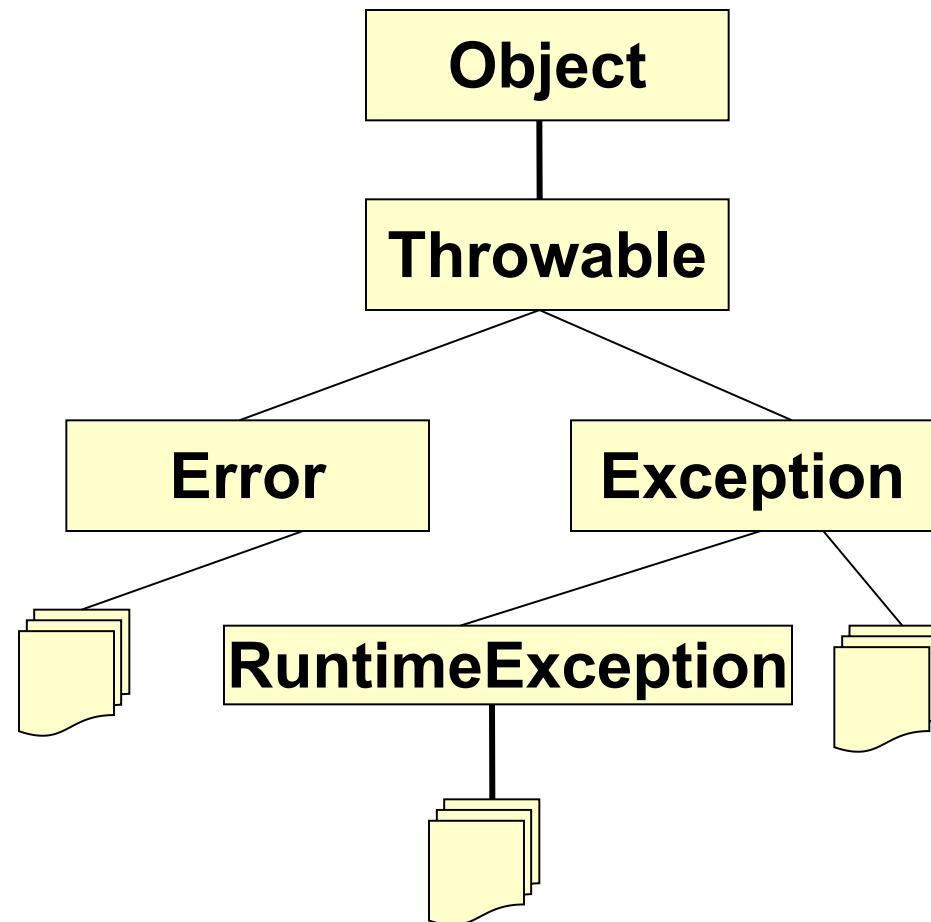
```
1. try {  
2.     // This is the first line of the “guarded region”.  
3. } catch(MyFirstError) {  
4.     // Put code here that handles this error.  
5. } catch(MySecondError) {  
6.     // Put code here that handles this error.  
7. } finally {  
8.     // Put code here to release any resource we  
9.     // allocated in the try clause.  
10. } // More code here
```



Exceptions call stack

- ◆ **Exception propagation**
- ◆ **An exception will cause your application to stop if it is never caught**
- ◆ **Frequently used Exception API**
 - ∅ **getCause()**
 - ∅ **getMessage()**
 - ∅ **printStackTrace()**

- ◆ Classes derive from *Error* represent unusual situations
- ◆ Applications won't be able to recover from an error
- ◆ All Java's exception types derive from *Exception*
- ◆ *RuntimeException*, also called unchecked exception
- ◆ A unchecked exception does not have to be caught
- ◆ Other exceptions, also called checked exceptions, must be treated



Common Unchecked Exceptions

- ◆ **java.lang.ArithmetricException**
- ◆ **java.lang.NullPointerException**
- ◆ **java.lang.ArrayIndexoutofBoundsException**
- ◆ **java.lang.NumberFormatException**
- ◆ **java.lang.SecurityException**
- ◆ **Java.lang.NegativeArraySizeException**

- ◆ **Handling: *try-catch-finally***
- ◆ **Using *throws* statement to declare that the code will cause exceptions**

```
public class JBeesLocalFile extends java.lang.Object {  
    private RandomAccessFile localFile;  
  
    /** Creates new JBeesLocalFile */  
    public JBeesLocalFile(String localFileName) throws IOException {  
        File f = new File(localFileName);  
        if(f.getParent() != null){  
            File dir = new File(f.getParent());  
            if(!dir.exists())  
                dir.mkdirs();  
        }  
        localFile = new RandomAccessFile(localFileName, "rw");  
    }  
  
    synchronized public void writeBytes(byte[] in, long offset) throws IOException {  
        localFile.seek(offset);  
        localFile.write(in);  
    }  
  
    public void close() throws IOException{  
        localFile.close();  
    }  
}
```

JBeesLocalFile.java

Writing Your Own Exceptions

◆ Deriving from Exception

```
public class MalformedB64EncodeException extends java.lang.Exception {  
  
    /**  
     * Creates new <code>MalformedB64EncodeException</code> without detail message.  
     */  
    public MalformedB64EncodeException() {  
        this("Error encoded string");  
    }  
  
    /**  
     * Constructs an <code>MalformedB64EncodeException</code> with the  
     * specified detail message.  
     * @param msg the detail message.  
     */  
    public MalformedB64EncodeException(String msg) {  
        super(msg);  
    }  
}
```

Throwing Your Own Exceptions

```
public class JBeesBase64 extends Object {  
    private final byte[] enc = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz  
    0123456789+/.getBytes();  
    private final byte pad = (byte)'=';  
    ...  
    public byte[] decode(byte[] in) throws MalformedB64EncodeException {  
        if(in == null) return null;  
        int len = in.length;  
        int loop = len / 4;  
        int remainder = len % 4;  
        if(remainder != 0) throw new MalformedB64EncodeException();  
        byte[] out;  
        if(in[len - 2] == pad) out = new byte[loop * 3 - 2];  
        else if(in[len - 1] == pad) out = new byte[loop * 3 - 1];  
        else out = new byte[loop * 3];  
  
        for(int i = 0; i < loop; i++) {  
            byte index0 = getIndex(in[i * 4]);  
            byte index1 = getIndex(in[i * 4 + 1]);  
            if(index0 == -1 || index1 == -1) throw new MalformedB64EncodeException();  
            out[i * 3] = (byte)((index0 << 2) | (index1 >> 4));  
            if(in[i * 4 + 2] == pad) break;  
            else{  
                index0 = getIndex(in[i * 4 + 2]);  
                if(index0 == -1) throw new MalformedB64EncodeException();  
                out[i * 3 + 1] = (byte)((index1 << 4) | (index0 >> 2));  
            }  
            if(in[i * 4 + 3] == pad) break;  
            else {  
                index1 = getIndex(in[i * 4 + 3]);  
                if(index1 == -1) throw new MalformedB64EncodeException();  
                out[i * 3 + 2] = (byte)((index0 << 6) | index1);  
            }  
        }  
        return out;  
    }  
}
```

- ◆ Checks a boolean-typed expression that a developer specifically proclaims must be true during program runtime execution
- ◆ Declare assertions with a new Java language keyword, assert
 - ∅ assert boolean-typed expression;
 - ∅ assert boolean-typed expression : string message to assertion facility;
- ◆ If the result of the boolean-typed expression is false, a java.lang.AssertionError is thrown

An Assertion Sample

```
public class AssertionTest{  
    public static void main(String[] args){  
        assert args.length == 1 :  
            "no. of command line arguments must be 1";  
        System.out.println(args[0]);  
    }  
}
```

u Compile AssertionTest using jdk 1.4

\$java –source 1.4 AssertionTest.java

u Run AssertionTest and enable assertions

\$java –enableassertions AssertionTest hello

Output: hello

\$java –enableassertions AssertionTest

Output: Exception in thread "main" java.lang.AssertionError: no. of command
lines must be 1 at AssertionTest.main(AssertionTest.java:4)



◆ In this chapter, we have discussed :

- u How to define exceptions
- u How to using try, catch, finally, throw and throws statements
- u Exception categories
- u Exceptions handing

Review Questions

- 1. (Level 1)What is the difference between error and exception?**
- 2. (Level 1)How to declare and handle exception?**
- 3. (Level 2)Which one of the following fragments shows the most appropriate way to throw an exception?**
 - a) Exception e= new IOException("File not found");
if(if.exists()){throw e;}**
 - b) if(if.exists()){throw new IOException("File not found");}**
 - c) if(if.exists()){throw IOException;}**
 - d) if(if.exists()){throw"File not found";}**
 - e) if(if.exists()){throw new IOException();}**

Assignment

- 1. (Level 1) Write three classes: OwnException, OwnExceptionSource and OwnExceptionHandler. OwnExceptionSource has a method a () which throws OwnException, and OwnExceptionHandler will call a () and handles exception .**
- 2. (Level 2) Write a class called DivionByZero2. There is a method called division () where you try to divide 10 by 0. Use exception concept to handle this case.**
- 3. (Level 2) Review , Compile and run the Base64 encoding/decoding sample class called JBeesBase64.java. You will have to add your own method called getIndex().**



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 4 Building GUIs

Brighten Your Way And Raise You Up.

Goals

- ◆ **java.awt.* and components**
- ◆ **Container, component and layout manager**
- ◆ **Using layout manager**
 - ∅ **BorderLayout**
 - ∅ **FlowLayout**
 - ∅ **GridLayout**
 - ∅ **GridBagLayout, etc.**
- ◆ **Adding components in container**
- ◆ **Using nested layout manager**
- ◆ **Key features of Java Foundation Classes(JFC)**

- ◆ **Abstract Window Toolkit**
- ◆ **Provides an arsenal of objects to construct graphical user interfaces (GUIs)**
- ◆ **Three main types of classes within the java.awt package**
 - ∅ **Component: The base class for the visual AWT classes, including Container**
 - ∅ **Container: A specialized component that holds other components**
 - ∅ **LayoutManager: An interface responsible for sizing and positioning components within a container**

Package java.awt

java.lang.Object

Border Layout
 Card Layout
 Checkbox Group
 Color
 Dimension
 Event
 Font
 Flow Layout
 FontMetrics
 Graphics
 GridBagConstraints
 GridBag layout.
 GridLayout
 Image
 Insets
 Point
 Polygon
 Rectangle
 Toolkit
 MenuComponent
Component

MenuBar
MenuItem

Menu-Popup Menu
CheckboxMenuItem

Button
 Canvas
 Checkbox
 Choice
 Container
 Label
 List
 Scrollbar
 TextComponent

Panel
Window
ScrollPane

TextArea
TextField

Applet(java.applet package)

Dialog-----**FileDialog**
Frame

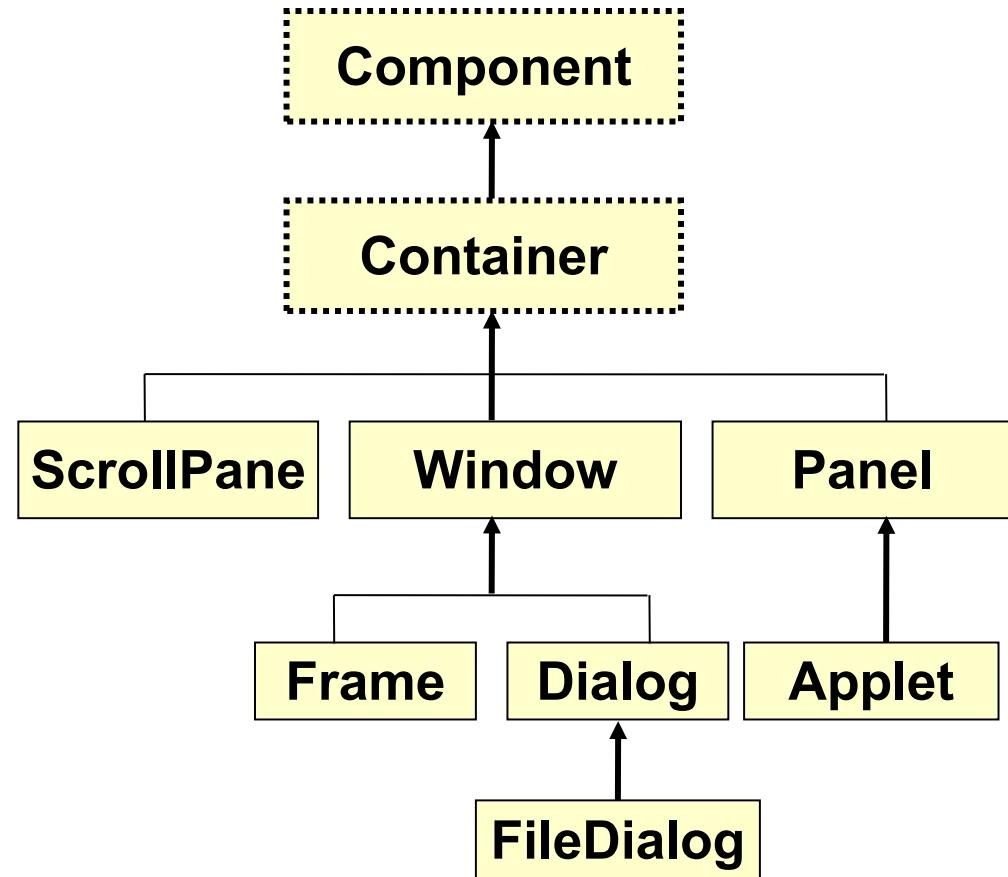
Exceptions-AWTException

Errors-AWTError

- ◆ Components can be added in container

- ◆ Usually using Frame, Dialog, Applet, Panel

- ◆ Can be extended



briup

- ◆ **add()**
add(Component com)
add(Component, int index)
add(Component component, Object constraints)
add(String name, Component component)
add(Component component, Object constraints, int index)
- ◆ **setLayout() and getLayout()**
- ◆ **getComponent()**
- ◆ **getInsets()**
- ◆ **remove and removeAll()**
- ◆ **validate()**

- ◆ Using container method `setLayout()` to set a layout manager
- ◆ Limiting the size and location of components within the container
- ◆ The methods `setLocation()`, `setSize()`, `setBounds()` on components define the size and position of components
- ◆ The default layout manager for *Window* type is `BorderLayout` manager
- ◆ The default layout manager for *Panel* type is `FlowLayout` manager

◆ Creating components

```
Button btn = new Button("Add");
```

```
Label lbl = new Label("Name");
```

...

◆ Adding components in the container using container method add()

```
add(btn);
```

```
add(lbl);
```



- ◆ Using component methods `addXXXListener()`

`btn.addActionListener(...)`



package sample;

```
public class MyFrame extends java.awt.Frame{  
    // step 1. Choose a container  
    private java.awt.Label southLabel;  
    private java.awt.Button northButton;  
    private java.awt.TextField eastTextField;  
    private java.awt.TextArea centerTextArea;  
    private java.awt.Label westLabel;  
  
    public static void main(String args[]) {  
        new MyFrame().show();  
    }  
  
    public MyFrame() {  
        initComponents();  
    }  
}
```



MyFrame.java(Cont.)

```
private void initComponents() {  
    //step 2. choose a layout manager: using the default—BorderLayout  
    //step 3. Initial andcomponents and add them to the container  
    centerTextArea = new java.awt.TextArea();  
    northButton = new java.awt.Button();  
    eastTextField = new java.awt.TextField();  
    southLabel = new java.awt.Label();  
    westLabel = new java.awt.Label();  
  
    setBackground(java.awt.Color.lightGray);  
    setCursor(new java.awt.Cursor(java.awt.Cursor.HAND_CURSOR));  
    setForeground(java.awt.Color.black);  
    setTitle("MyFrame V1.0");  
    addWindowListener(new java.awt.event.WindowAdapter() {  
        public void windowClosing(java.awt.event.WindowEvent evt) {  
            exitForm(evt);  
        }  
    });  
}
```



MyFrame.java(Cont.)

```
centerTextArea.setText("I am in the center");
add(centerTextArea, java.awt.BorderLayout.CENTER);
northButton.setActionCommand("I am in the north");
northButton.setLabel("I am in north");
//step 4. Create event handler
northButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        northButtonActionPerformed(evt);
    }
});
add(northButton, java.awt.BorderLayout.NORTH);
eastTextField.setText("I am in the east");
add(eastTextField, java.awt.BorderLayout.EAST);
southLabel.setAlignment(java.awt.Label.CENTER);
southLabel.setText("I am in the south");
add(southLabel, java.awt.BorderLayout.SOUTH);
```



MyFrame.java(Cont.)

```
westLabel.setText("I am in the west");
add(westLabel, java.awt.BorderLayout.WEST);

pack();
}

//event handling
private void northButtonActionPerformed(java.awt.event.ActionEvent evt) {
setVisible(false);
dispose();
System.exit(0);
}
private void exitForm(java.awt.event.WindowEvent evt) {setVisible(false);
dispose();
System.exit(0);
}
}
```



- ◆ Derived from the **Window** class
- ◆ Top-level container
- ◆ Having title, borders and menu bars
- ◆ The default layout manager is **BorderLayout** manager
- ◆ Using **setLayout()** to change layout manager
- ◆ Creating a frame: **Frame frame = new Frame("Title");**

- ◆ **Typically added to another container**
- ◆ **Used to group components within a container**
- ◆ **The default layout manager is FlowLayout manager**

- ◆ **Flow Layout**
- ◆ **Border Layout**
- ◆ **Grid Layout**
- ◆ **Card Layout**
- ◆ **GridBag Layout**

- ◆ The flow layout manager respects the preferred size of each component
- ◆ Constructors of flow layout manager

FlowLayout()

FlowLayout(int align)

FlowLayout(int align, int hgap, int vgap)

int align: `FlowLayout.LEFT`, `FlowLayout.RIGHT`,

`FlowLayout.CENTER`,`FlowLayout.LEADING`,

`FlowLayout.TRAILING`

A Flow Layout Sample

```
package sample;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class MyFlow {
    private Frame f;
    private Button button1, button2, button3;
    public static void main (String args[]) {
        MyFlow mflow = new MyFlow ();
        mflow.go();
    }
    public void go() {
        f = new Frame ("Flow Layout");
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent evt) {
                f.setVisible(false);
                f.dispose();
                System.exit(0);
            }
        });
        //f.setLayout(new FlowLayout());
        f.setLayout(new FlowLayout(FlowLayout.LEADING, 20, 20));
        button1 = new Button("Ok");
        button2 = new Button("Open");
        button3 = new Button("Close");
        f.add(button1);
        f.add(button2);
        f.add(button3);
        f.setSize (100,100);
        //f.pack();
        f.setVisible(true);
    }
}
```

Border Layout Manager

- ◆ Divides the container into 5 areas:
 - Ø **BorderLayout.CENTER**
 - Ø **BorderLayout.EAST**
 - Ø **BorderLayout.WEST**
 - Ø **BorderLayout.SOUTH**
 - Ø **BorderLayout.NORTH**
- ◆ The preferred height is maintained in South and North area, but the width is resized to fill the container width
- ◆ The preferred width is maintained in East and West area, but the height is resized to fill the container height
- ◆ In the center area, the width and height are resized
- ◆ Constructors:
 - Ø **BorderLayout()**
 - Ø **BorderLayout(int hgap, int vgap)**

A Border Layout Sample

```
package sample;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class MyBorder {
    Frame f;
    Button east, south, west, north, center;
    public static void main(String args[]) {
        MyBorder mb = new MyBorder();
        mb.go();
    }
    public void go() {
        f = new Frame("BorderLayout Example");
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent evt) {
                f.setVisible(false);
                f.dispose(); System.exit(0);
            }
        });
        f.setBounds(0, 0, 300, 300);
        f.setLayout(new BorderLayout());
        north = new Button("North");
        south = new Button("South");
        east = new Button("East");
        west = new Button("West");
        center = new Button("Center");
        f.add(BorderLayout.NORTH, north);
        f.add(BorderLayout.SOUTH, south);
        f.add(BorderLayout.EAST, east);
        f.add(BorderLayout.WEST, west);
        f.add(BorderLayout.CENTER, center);

        f.setVisible(true);
    }
}
```

- ◆ Arranging components in a grid within a container
- ◆ Each cell in the grid is of equal dimensions
- ◆ Ignoring the preferred size of components
- ◆ Constructors:
 - ∅ **GridLayout()**
 - ∅ **GridLayout(int rows, int cols)**
 - ∅ **GridLayout(int rows, int cols, int hgap, int vgap)**

A Grid Layout Sample

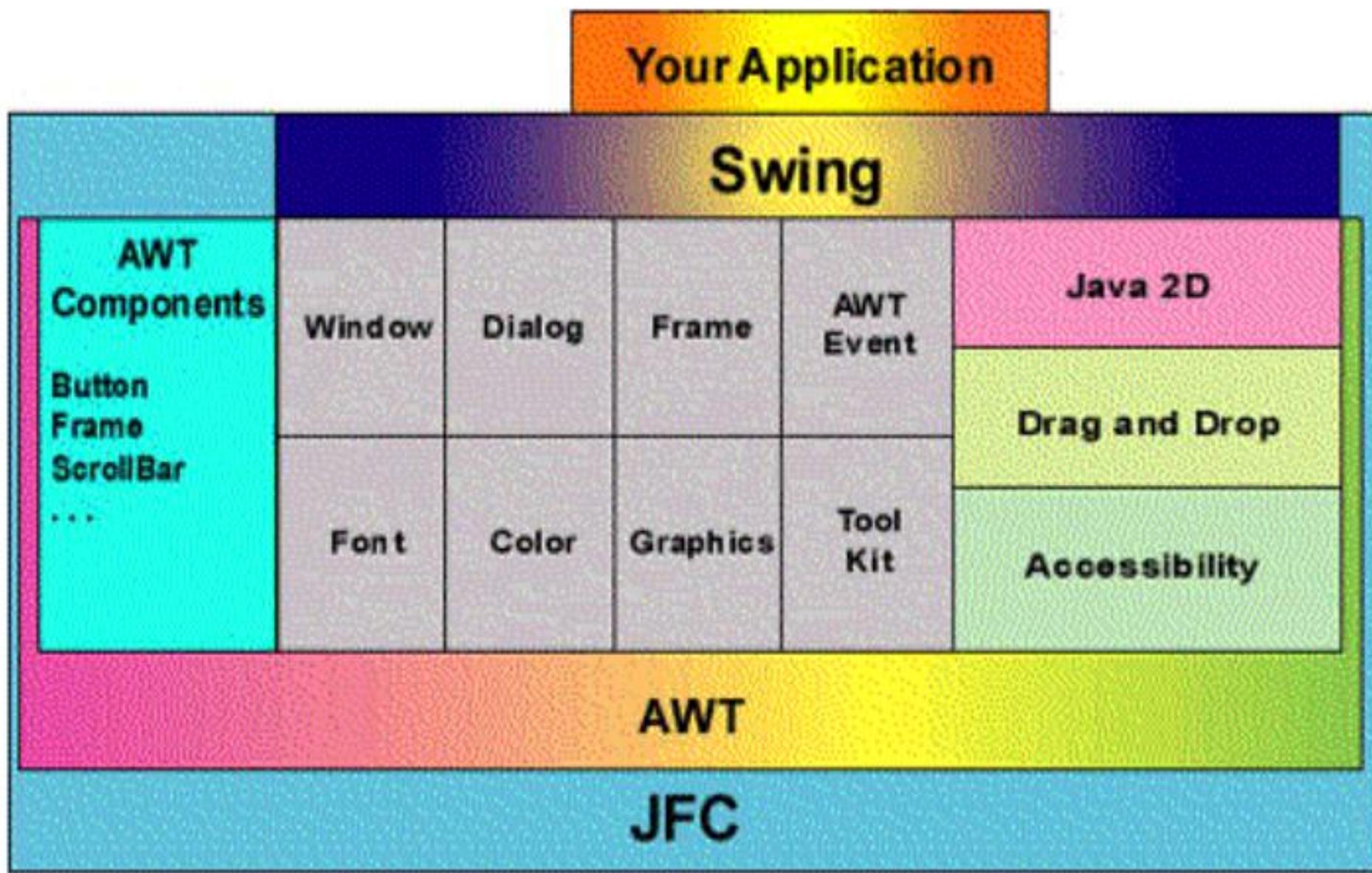
```
import java.awt.*;
import java.awt.event.*;
public class MyGrid {
    private Frame f;
    private Button[] btn;
    public static void main(String args[]) {
        MyGrid grid = new MyGrid();
        grid.go();
    }
    public void go() {
        f = new Frame("Grid example");
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent evt) {
                f.setVisible(false);
                f.dispose();
                System.exit(0);
            }
        });
        f.setLayout (new GridLayout (3, 3, 10, 10));
        btn = new Button[9];
        for(int i = 0; i < 9; i++) {
            int j = i + 1;
            btn[i] = new Button("'" + j);
            f.add(btn[i]);
        }
        f.pack();
        f.setVisible(true);
    }
}
```

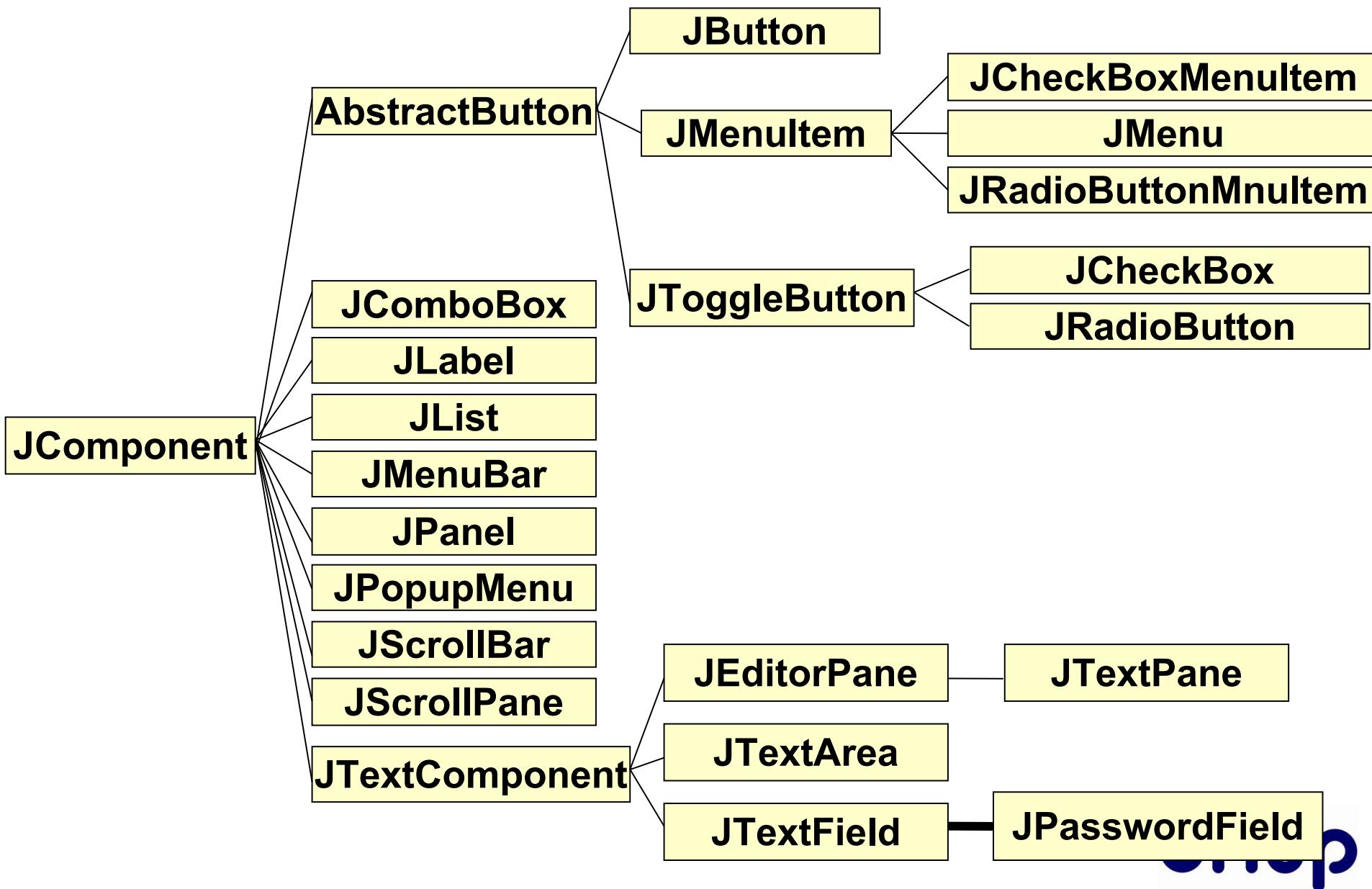
- ◆ Treating each component as a card, with only one card being visible at any given time
- ◆ Constructors
 - Ø CardLayout()
 - Ø CardLayout(int hgap, int vgap)
- ◆ Methods in CardLayout
 - Ø void first(Container c)
 - Ø void last(Container c)
 - Ø void next(Container c)
 - Ø void show(Container c, String name)

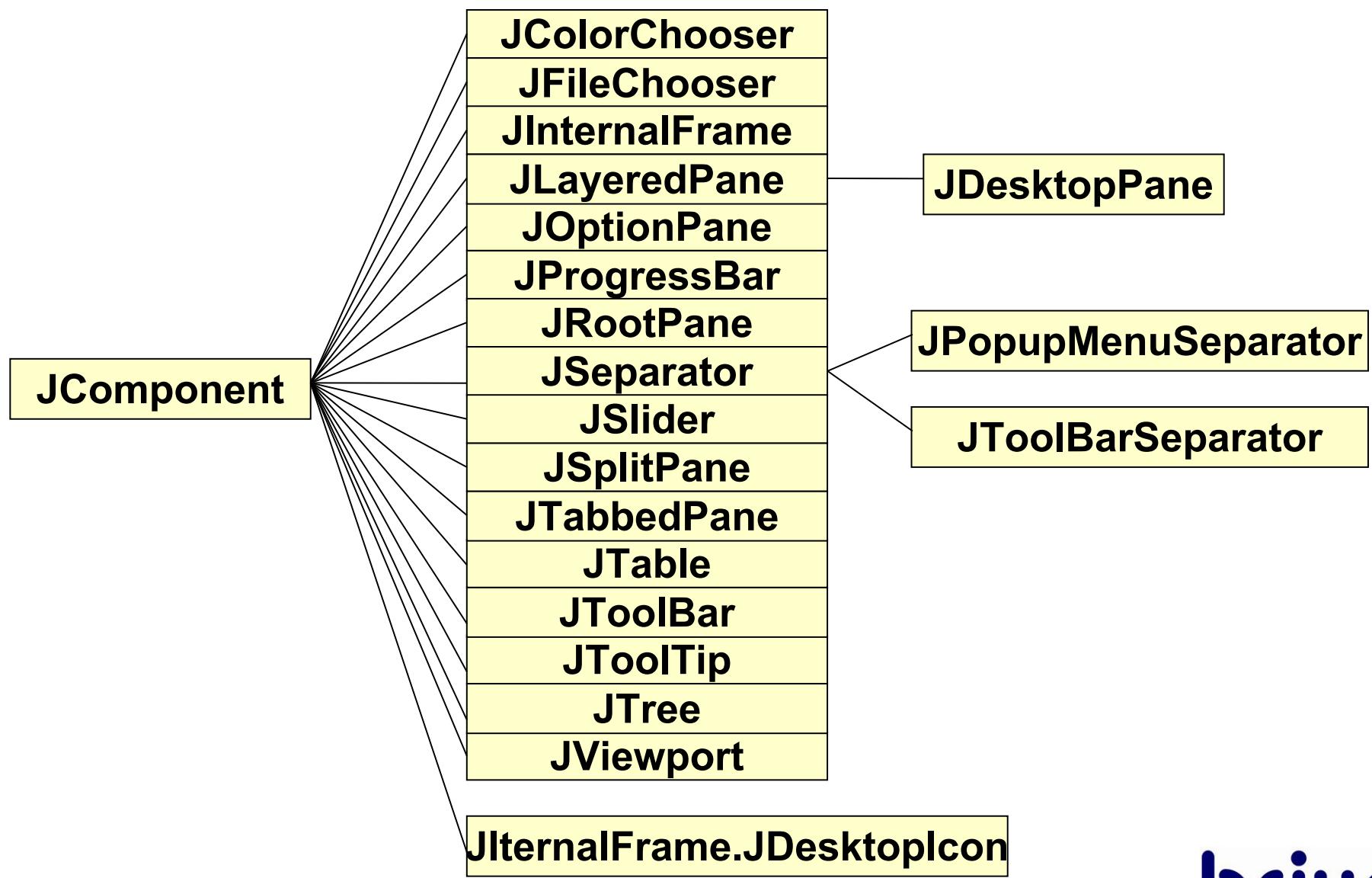
A Card Layout Sample

```
package sample;
import java.awt.*;
import java.awt.event.*;
public class MyCard {
    public static void main(String args[]) {
        new MyCard().go();
    }
    public void go() {
        final Frame f = new Frame("CardLayout Example");
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent evt) {
                f.setVisible(false);
                f.dispose();
                System.exit(0);
            }
        });
        f.setSize(300, 100);
        f.setLayout(new CardLayout());
        final Frame f1 = f;
        for(int i = 1; i <= 5; ++i) {
            Button b = new Button("Button " + i);
            b.setSize(100, 25);
            b.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent ae) {
                    CardLayout cl = (CardLayout)f1.getLayout();
                    cl.next(f1);
                }
            });
            f.add(b, "button" + i);
        }
        f.setVisible(true);
    }
}
```

- ◆ Leveraging Netscape, IBM and Lighthouse design to create a set of Graphical User Interface (GUI) classes
- ◆ Providing a more polished look and feel than the standard AWT component set
- ◆ JFC is composed of five APIs: AWT, Java 2D, Accessibility, Drag and Drop, and Swing
- ◆ Swing includes a component set that is targeted at forms-based applications
- ◆ Swing GUIs are visually presented, independent of platform
- ◆ The AWT 1.1 widgets and event model are still present for the Swing widgets







◆ In this chapter, we have discussed:

- u Java.awt package and components
- u Container, component and layout manager
- u How to use layout manager
 - Ø BorderLayout
 - Ø FlowLayout
 - Ø GridLayout
 - Ø GridBagLayout s etc.
- u How to add components in container
- u How to use nested layout manager

- 1. (Level 1)What are the four steps for building GUI?**
- 2. (Level 1)What are the five common layout managers?**
- 3. (Level 1)What is the difference between frame and panel?**
- 4. (Level 2)What is the default layout manager for frame and panel respectively?**

- 1. (Level 1)Write a class called AWTFrame. The frame uses border layout and contains a status label and panel which, in turn, contains two buttons. The panel use flow layout**

- 2. (Level 2)Write a class called AWTCCombinationFrame. The frame uses border layout and contains a status label and a panel which, in turn, uses grid layout and contains twelve buttons(number 1-9 and *, 0, #)**



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 5 **The AWT Event Model**

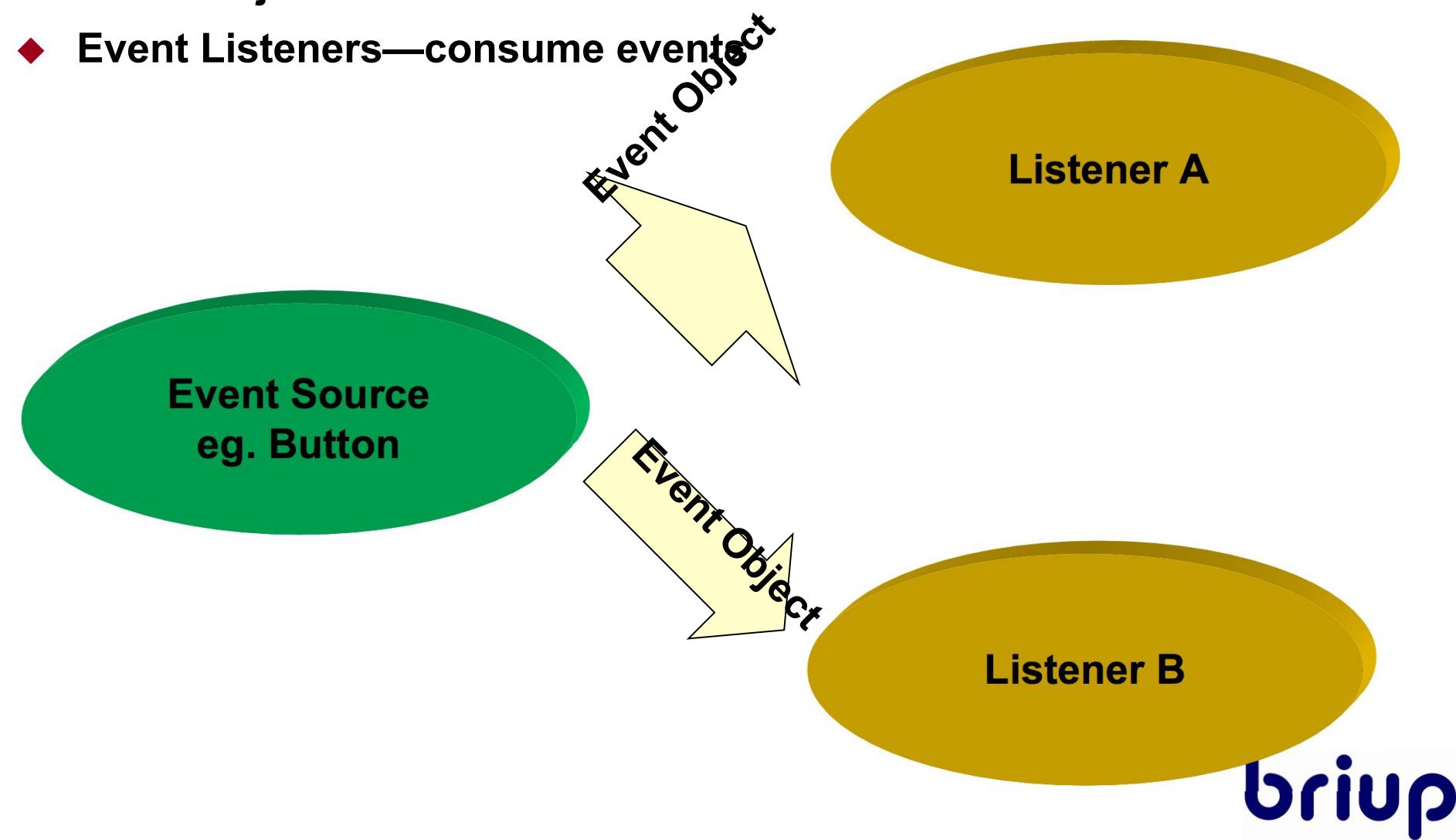
Brighten Your Way And Raise You Up.

Goals

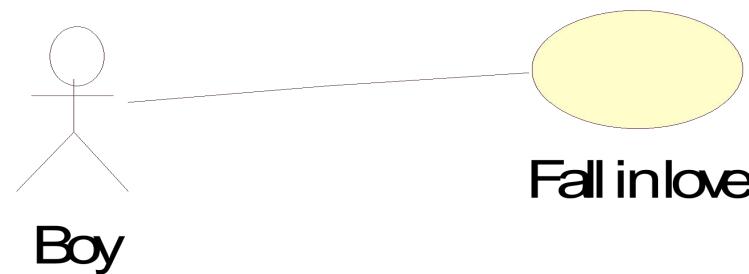
- ◆ Coding to handle AWT events
- ◆ The concept of adapter
- ◆ Actions related with the details of an event
- ◆ Handlers and interfaces for events

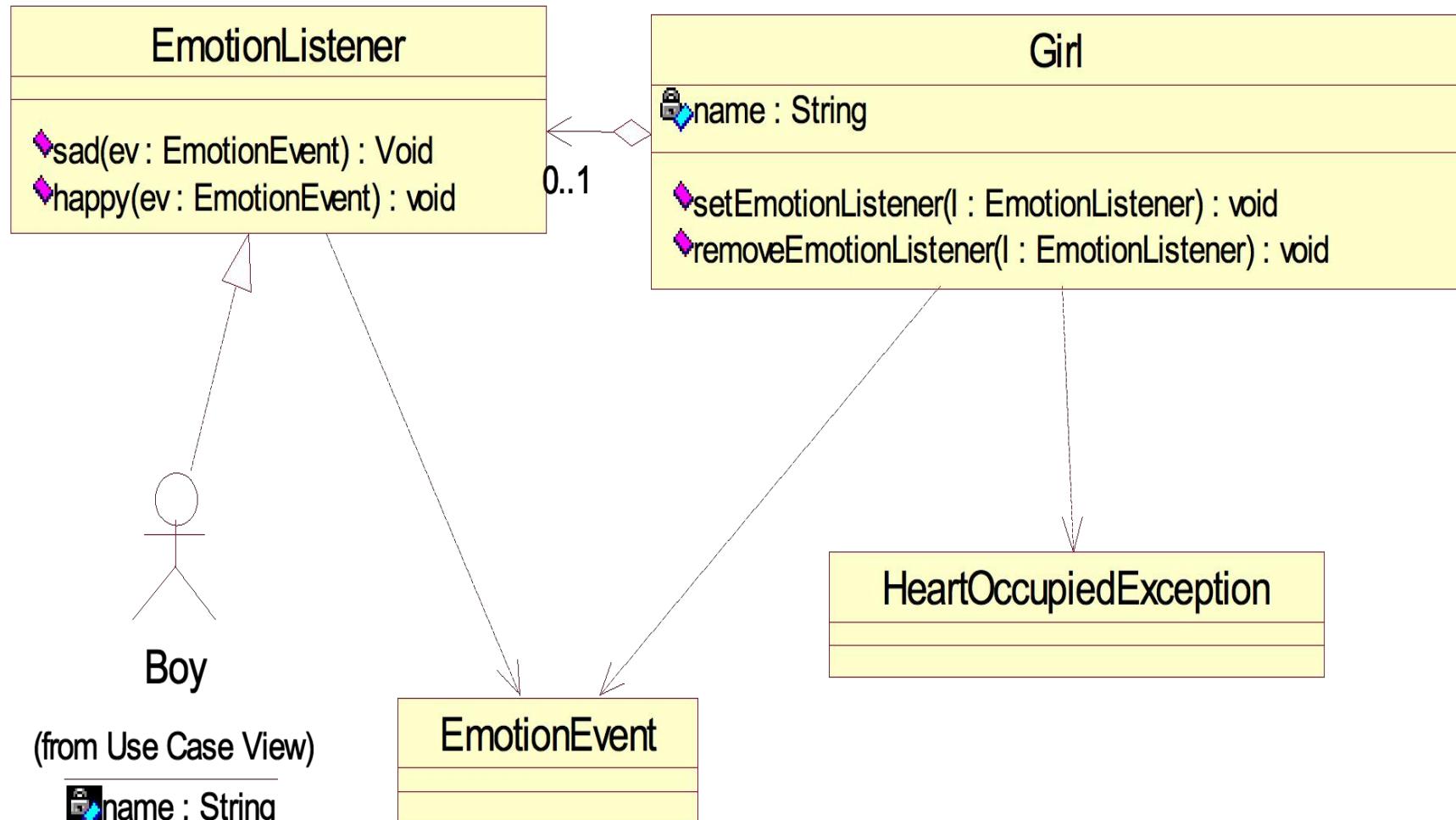
briup

- ◆ **Event Sources**—generate events
- ◆ **Event Objects**—describe events
- ◆ **Event Listeners**—consume events



- ◆ Event Sources—Girl
- ◆ Event Objects—EmotionEvent
- ◆ Event Listeners—EmotionListener(Boy is an example)





- ◆ **Embodying information related to a particular type of event**
- ◆ **At least, an event object contains a reference to the object that fire the event**
- ◆ **All events subclass java.util.EventObject**

```
public class EmotionEvent extends java.util.EventObject{  
    public EmotionEvent(Object src){  
        super(src);  
    }  
}
```

- ◆ An event source is a component or object that generates events
- ◆ Providing methods that allow other methods to add and remove event listeners
- ◆ Maintaining a list of interested event listeners
- ◆ Containing logic to create and deliver event objects of the appropriate type (class) to all interested event listeners

An Event Source--Girl

```

public class Girl extends Thread{
    private EmotionListener listener;
    private String name;
    private int day;

    public Girl (String lover){
        this.name = lover;
    }

    public String getLover() { return lover; }

    public void setEmotionListener(
        EmotionListener l)
        throws HeartOccupiedException{
        if(listener != null) {
            System.out.println("Do not you know I
have had a lover");
            throw new HeartOccupiedException("My
heart is full");
        }

        System.out.println("Finally I have another
half");
        listener = l;
    }

    public void
        removeEmotionListener(EmotionListener l){
        if(listener == l){
            System.out.println("I can't live without
you");
            listener = null;
        }
    }
}

```

```

public void run(){
    while(true){
        try{
            Thread.sleep(1000);
            day++;
            fire();
        }catch(Exception e){}
    }

    private void fire (){
        EmotionEvent evt = new EmotionEvent(this);
        if(listener == null){
            System.out.println("Being a single is lone.");
            return;
        }
        if(day % 2 == 0) listener.happy(evt);
        else listener.sad(evt);
    }
}

```

- ◆ An interface represents objects to be notified of an event
- ◆ Derived from `java.util.EventListener`

```
public interface EmotionListener extends java.util.EventListener{  
    public void happy(EmotionEvent evt);  
    public void sad(EmotionEvent evt);  
}
```

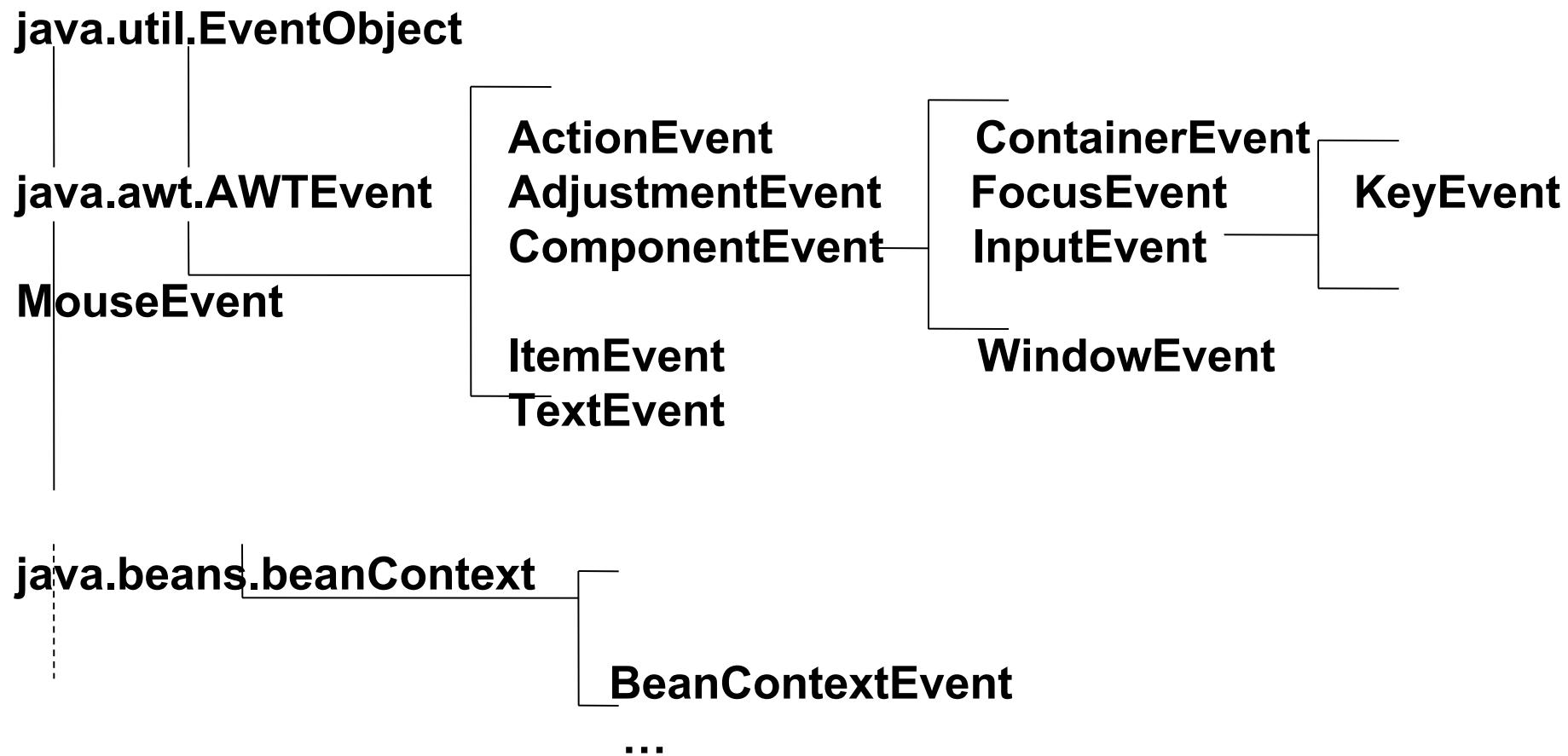


A Concrete Event Listener—Boy

```
public class Boy implements EmotionListener{
    private String name;
    public Boy(String name){ this.name = name;}
    public void happy(EmotionEvent evt){
        String lover = ((Girl)evt.getSource()).getLover();
        System.out.println(lover + ", I am also happy");
    }
    public void sad(EmotionEvent evt){
        String lover = ((Girl)evt.getSource()).getLover();
        System.out.println(lover + ", take care, you make me sad too.");
    }
    public static void main(String[] args) throws HeartOccupiedException{
        GirlBoyFriend lover = new Girl ("Mary");
        Boy me = new Boy("larry");
        lover.setEmotionListener(me);
        lover.start();
    }
}
```

```
public class HeartOccupiedException extends Exception{  
    public HeartOccupiedException(){  
        this("No reason");  
    }  
  
    public HeartOccupiedException(String mesg){  
        super(mesg);  
    }  
}
```





- ◆ An item is double-clicked in a List component

- A Button is selected
- A MenuItem is selected
- The ENTER key is pressed in a TextField component

- ◆ **Text has changed in a TextField component**
- ◆ **Text has changed in a TextArea component**

- ◆ A check box is selected or cleared
- ◆ A CheckBoxMenuItem is selected or cleared
- ◆ An item is selected in a Choice component
- ◆ One or more items is selected in a List component



Category	Interface Name	Methods
Action	ActionListener	actionPerformed(ActionEvent)
Item	ItemListener	itemStateChanged(ItemEvent)
Mouse motion	MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
Mouse button	MouseListener	mousePressed(MouseEvent) MouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mouseClicked(MouseEvent)
Key	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
Focus	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
Adjustment	AdjustmentListener	adjustmentValueChanged (AdjustmentEvent)
Component	ComponentListener	componentMoved(ComponentEvent) componentHidden(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)



Category	Interface Name	Methods
Window	WindowListener	WindowClosing(WindowEvent) windowOpened(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent)
Container	ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
Text	TextListener	textValueChanged(TextEvent)

```
package sample;  
import java.awt.*;  
import java.awt.event.*;  
  
public class MultipleListen implements  
MouseMotionListener,  
MouseListener, WindowListener {  
    private Frame f;  
    private TextField tf;  
  
    public static void main(String args[]) {  
        MultipleListen two = new MultipleListen();  
        two.go();  
    }  
}
```

```
public void go() {  
    f = new Frame("Two listeners example");  
    f.setBackground(Color.green);  
    f.addWindowListener(this);  
    Label label = new Label ("Click and drag the mouse");  
    label.setBackground(Color.white);  
    f.add (label, BorderLayout.NORTH);  
    tf = new TextField (30);  
    f.add (tf, BorderLayout.SOUTH);  
    f.addMouseListenerMotionListener(this);  
    f.addMouseListener (this);  
    f.setSize(300, 200);  
    f.setVisible(true);  
}
```

```
// These are MouseMotionListener events
public void mouseDragged (MouseEvent e) {
    String s = "Mouse dragging: X = " + e.getX() + " Y = " + e.getY();
    tf.setText (s);
}

public void mouseMoved (MouseEvent e) {}

// These are MouseListener events
public void mouseClicked (MouseEvent e) {}
public void mouseEntered (MouseEvent e) {
    String s = "The mouse entered";
    tf.setText (s);
}

public void mouseExited (MouseEvent e) {
    String s = "The mouse has left the building";
    tf.setText (s);
}
```



```
public void mousePressed (MouseEvent e) {}  
public void mouseReleased (MouseEvent e) {}  
//These are WindowListener events  
public void windowActivated(java.awt.event.WindowEvent windowEvent) {}  
public void windowClosed(java.awt.event.WindowEvent windowEvent) {}  
public void windowClosing(java.awt.event.WindowEvent windowEvent) {  
    f.setVisible(false);  
    f.dispose();  
    System.exit(0);  
}  
public void windowDeactivated(java.awt.event.WindowEvent windowEvent) {}  
public void windowDeiconified(java.awt.event.WindowEvent windowEvent) {}  
public void windowIconified(java.awt.event.WindowEvent windowEvent) {}  
public void windowOpened(java.awt.event.WindowEvent windowEvent) {}  
}
```

- ◆ An event adapter **Implements A particular listener with empty implementations**
- ◆ You just subclass an adapter and override the methods you are interested in

◆ In this chapter, we have discussed:

- u Coding to handle AWT events
- u The concept of adapter
- u Actions related with the details of an event
- u Handlers and interfaces ofr events

- 1. (Level 1)What are the three concepts in event handling?**
- 2. (Level 2)What is an event adapter and why we need it?**
- 3. (Level 2)What is an item event and which cases can trigger it?**



- 1. (Level 1)Write a class called AWTEventFrame to modify the AWTFrame class .Add event handler for the two buttons using inner class and windows adapter.**
- 2. (Level 2)Write a class called Resume to modify the ResumeFrame.java class to add event handlers.**
- 3. (Level 3)Write a class called MyAWTCalculator to simulate a real one to add, substract, multiply and divide numbers.**



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 6

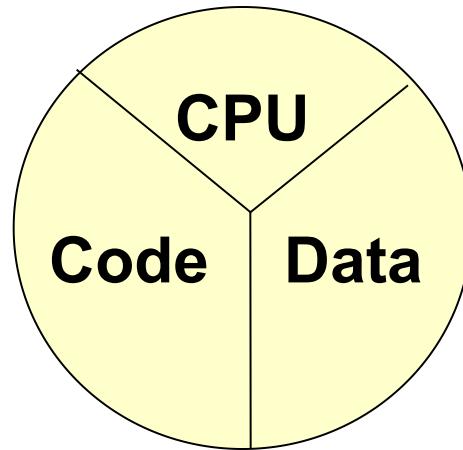
Threads

Brighten Your Way And Raise You Up.

- ◆ **The concept of threads**
- ◆ **Creating, instantiating, and starting new threads**
- ◆ **Transitioning between thread states**
- ◆ **Concurrent access problems and synchronized threads**
- ◆ **Communicating with objects by waiting and notifying**
- ◆ **Deadlocked threads**
- ◆ **Scheduling threads using priority and yielding**

- ◆ A *thread* is a single sequential flow of control within a program
- ◆ A software *engine* that can process a sequence of instructions
- ◆ Threads run at the same time, independently of one another
- ◆ `java.lang.Thread`

- ◆ Processor
- ◆ Code
- ◆ Data



A thread or
execution context

- ◆ When a thread starting, code in method “*public void run()*” executed
- ◆ Two ways of creating java threads
 - ∅ extending the class Thread
 - ∅ implementing the interface Runnable

◆ Defining

```
public class MyThread extends Thread {  
    public void run() {...}  
}
```

◆ Instantiating

```
MyThread testThread = new MyThread()
```



◆ Constructors

- ∅ **Thread()**
- ∅ **Thread(Runnable target)**

◆ Methods

- ∅ **run()**
- ∅ **start()**

◆ Defining

```
public class MyRunnable implements Runnable {  
    public void run() {...}  
}
```

◆ Instantiating

```
MyRunnable testRunnable = new MyRunnable();
```

```
Thread testThread = new Thread(testRunnable)
```

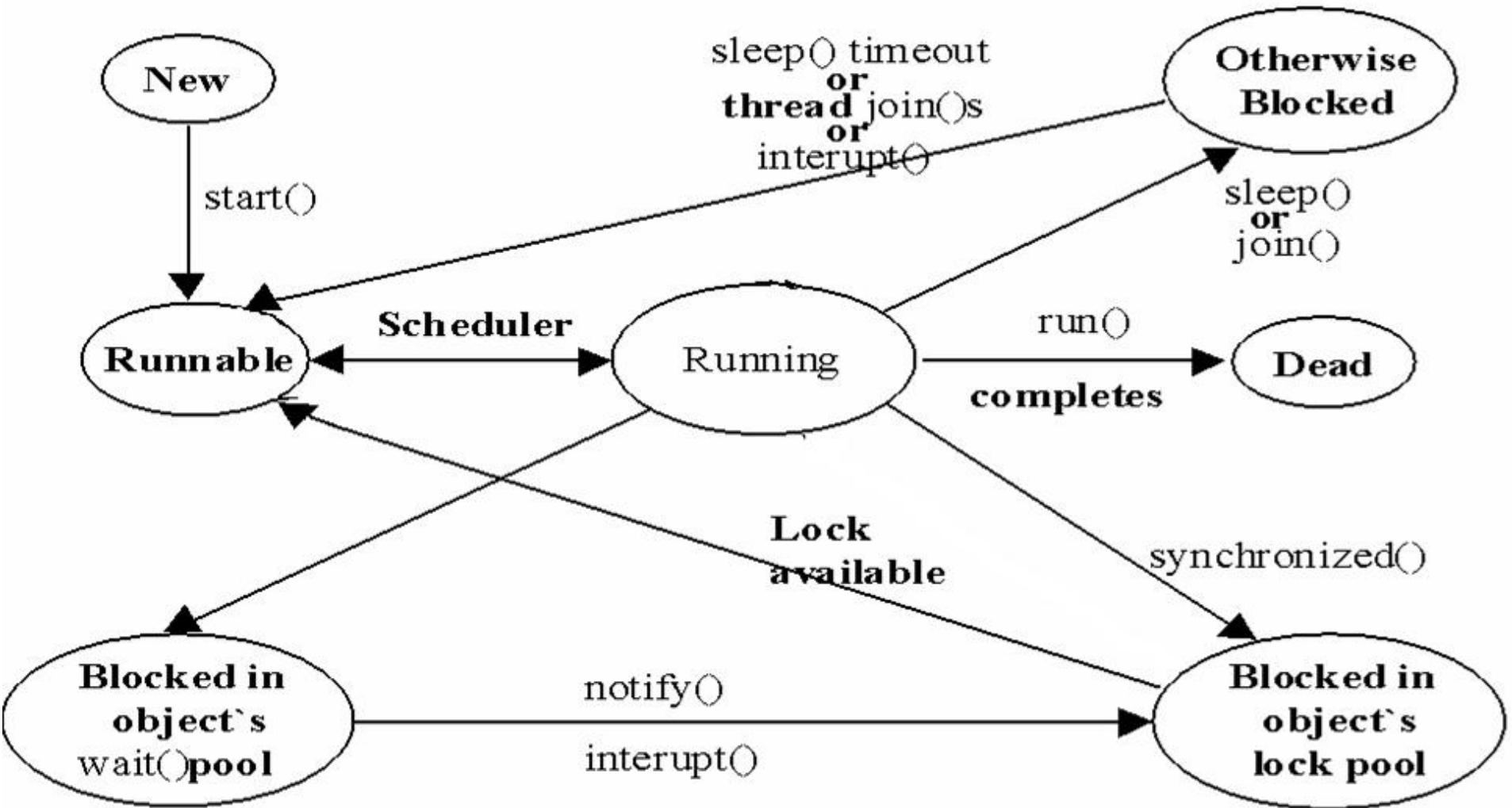
- ◆ run()

briup

- ◆ **testThread.start()**
- ◆ **Executing code in method run()**

- ◆ Extending the class *Thread* is Object-Oriented programming
- ◆ Implementing the interface *Runnable* solves single inheritance limitation

Thread states



- ◆ Using tag variables to indicate to terminate a thread
- ◆ Method `stop()` in the class `Thread` is deprecated
- ◆ The static method `Thread.currentThread()` return the current reference of a thread

```
public class MyThread extends Thread{  
    private boolean stop; //tag variable  
    public void setStop(boolean stop){ this.stop = stop;}//set tag value  
    public void run{  
        while(!stop){...}  
    }  
}
```

- ◆ Judging threads status

isAlive()

briup

Blocking Threads to Run

- ◆ **sleep()**

Thread.sleep(5*1000)

- ◆ **Interrupting a thread**

sleepingThread.interrupt()

- ◆ **Checking interrupted status**

otherThread.isInterrupted()

Thread.interrupted() :

interrupted status of the thread is cleared

- ◆ **Blocking threads**

otherThread.join()



- ◆ The threads often share information in some degree
- ◆ There is no guarantee that two lines of code will be executed one after the other by the CPU
- ◆ Java uses *synchronized* mechanism to control concurrent access problem

Keyword synchronized

- ◆ **Marking a group of variables that should have exclusive use for a thread**
- ◆ **Synchronized methods**

synchronized void methodA() {...}

- ◆ **Synchronized codes**

void methodB() {

...

synchronized(this) {...}

...

}



A Concurrency Control Example—SynchBankTest

```
public class SynchBankTest {  
    public static void main(String[] args) {  
        Bank b = new Bank(NACCOUNTS, INITIAL_BALANCE);  
        int i;  
        for (i = 0; i < NACCOUNTS; i++) {  
            TransferThread t = new TransferThread(b, i,  
                INITIAL_BALANCE);  
            t.setPriority(Thread.NORM_PRIORITY + i * 2);  
            t.start();  
        }  
    }  
  
    public static final int NACCOUNTS = 10;  
    public static final int INITIAL_BALANCE = 10000;  
}
```



```
class Bank {  
    public Bank(int n, int initialBalance)  {  
        accounts = new int[n];  
        for (int i = 0; i < accounts.length; i++) accounts[i] = initialBalance;  
        ntransacts = 0;  
    }  
  
    public synchronized void transfer(int from, int to, int amount)  
        throws InterruptedException {  
        while (accounts[from] < amount)  wait();  
        accounts[from] -= amount; accounts[to] += amount; ntransacts++; notifyAll();  
        if (ntransacts $ NTEST == 0) test();  
    }  
  
    public synchronized void test() {  
        int sum = 0;  
        for (int i = 0; i < accounts.length; i++) sum += accounts[i];  
        System.out.println("Transactions:" + ntransacts + " Sum: " + sum);  
    }  
}
```

```
public int size() {    return accounts.length; }
```

```
public static final int NTEST = 10000;
```

```
private final int[] accounts;
```

```
private long ntransacts = 0;
```

```
static final int INITIAL_BALANCE = 10000;
```

```
}
```



A Concurrency Control Example—SynchBankTest(Cont.)

```
class TransferThread extends Thread {  
    public TransferThread(Bank b, int from, int max) {  
        bank = b; fromAccount = from; maxAmount = max;  
    }  
  
    public void run(){  
        try{  
            while (!interrupted()) {  
                int toAccount = (int)(bank.size() * Math.random());  
                int amount = (int)(maxAmount * Math.random());  
                bank.transfer(fromAccount, toAccount, amount);  
                sleep(1);  
            }  
        } catch(InterruptedException e) {}  
    }  
  
    private Bank bank;  
    private int fromAccount;  
    private int maxAmount;  
}
```



- ◆ The signaling system available to Java threads to ensure the proper order of execution between them
- ◆ Using `wait()` and `notify()` by two independent threads
- ◆ Using `notifyAll()` when many threads may be waiting

WaitTest.java

```
public class WaitTest {  
    public static void main(String [] args) {  
        ThreadB b = new ThreadB();  
        b.start();  
        System.out.println("Total is: " + b.getTotal());  
    }  
}  
class ThreadB extends Thread {  
    int total;  
    public void run() {  
        synchronized(this) {  
            for(int i=0;i<10000;i++) { total += i; try{Thread.sleep(500);}catch(Exception e){}}  
            notify();  
        }  
    }  
  
    synchronized public int getTotal() {  
        try{ wait(); }catch(InterruptedException e) {} //comment out this line and run  
again  
        return total;  
    }  
}
```

- ◆ Thread starvation or deadly embrace

```
1. public class DeadlockRisk {  
2.     private static class Resource {  
3.         public int value;  
4.     }  
5.     private Resource resourceA = new Resource();  
6.     private Resource resourceB = new Resource();  
7.     public int read() {  
8.         synchronized(resourceA) { // May deadlock here  
9.             synchronized(resourceB) {  
10.                 return resourceB.value + resourceA.value;  
11.             }  
12.         }  
13.     }  
14.  
15.     public void write(int a, int b) {  
16.         synchronized(resourceB) { // May deadlock here  
17.             synchronized(resourceA) {  
18.                 resourceA.value = a;  
19.                 resourceB.value = b;  
20.             }  
21.         }  
22.     }  
23. }
```

- ◆ **The current thread will always lose its turn when the yield() method is called**

Thread.yield()

- ◆ **specify the priorities of threads**

setPriority()



- ◆ **stop()**
- ◆ **resume()**
- ◆ **suspend()**

Review questions

- 1. (Level 1)What are the differences between two ways to create threads?**
- 2. (Level 1)What are the four states for threads?**
- 3. (Level 1)How to solve the concurrent access problem for threads?**
- 4. (Level 2)What does deadlock means?**
- 5. (Level 2) A thread wants to make a second thread ineligible for execution. To do this, the first thread can call the yield() method on the second thread?**
A. true B.false
- 6. (Level 2)A java monitor must either extend Thread or implement Runnable.**
A. true B.false

Review Question – Cont.

7. (Level 3) A thread's run() method includes the following lines:

```
1. try{  
2. sleep(100);  
3. }catch(InterruptedException e){}
```

Assuming the thread is not interrupted, which statement is true?

- a) The code will not compile, because exception can not be caught in a thread's run() method
- b) At line 2, the thread will stop running. Exception will resume in, at most, 100 milliseconds
- c) At line 2, the thread will stop running, Exception will resume in exactly 100 milliseconds.
- d) At line 2, the thread will stop running, Execution will resume some time after 100 milliseconds have elapsed.

Assignment

1. (Level 1) Write a class called ThreadExample which will run another class called SimpleRunnable which implements Runnable
2. (Level 2) Write a class called Race. It simulates the race between rabbit and turtle. Use Math.random() to make the competition more interesting.
3. (Level 2) Write three classes called Counter, Printer and Storage. The Storage class should store an integer. The Counter class should create a thread that starts counting from 0 and stores each value in the Storage class. The Printer class should create a thread that keeps reading the value in the Storage class and printing it. It should be ensured that each number is printed exactly once, by adding suitable synchronization.
4. (Level 3) Write a class called StackTest which implements Last in First Out data structure and is thread safe.



杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

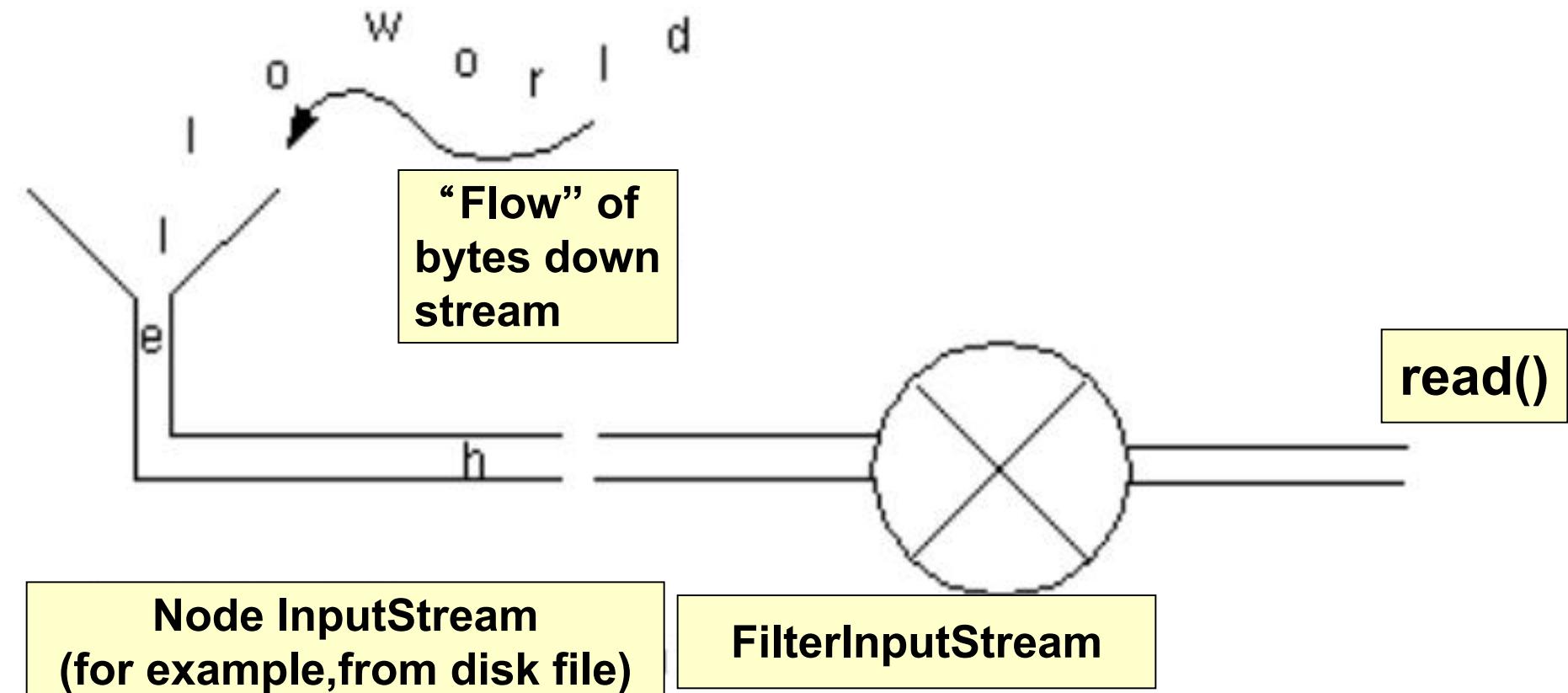
Module 7 Stream I/O and Files

Brighten Your Way And Raise You Up.

Goals

- ◆ **Navigating a file system**
- ◆ **The java.io hierarchy**
- ◆ **Creating files and file-choosers**
- ◆ **Choosing suitable reader and/or writer**
- ◆ **Operations on files and directories**
- ◆ **Reading, writing and updating data from/to files**
- ◆ **Using serialization mechanism**

- ◆ Byte sources or destinations
- ◆ Two kinds of stream, InputStream and OutputStream
-



InputStream Methods

◆ Three *read()* methods

- Ø **int read()**
- Ø **int read(byte[])**
- Ø **int read(byte[], int, int)**

◆ Other methods

- Ø **void close()**
- Ø **int available()**
- Ø **skip(long)**
- Ø **boolean markSupported()**
- Ø **void mark(int)**
- Ø **void reset()**

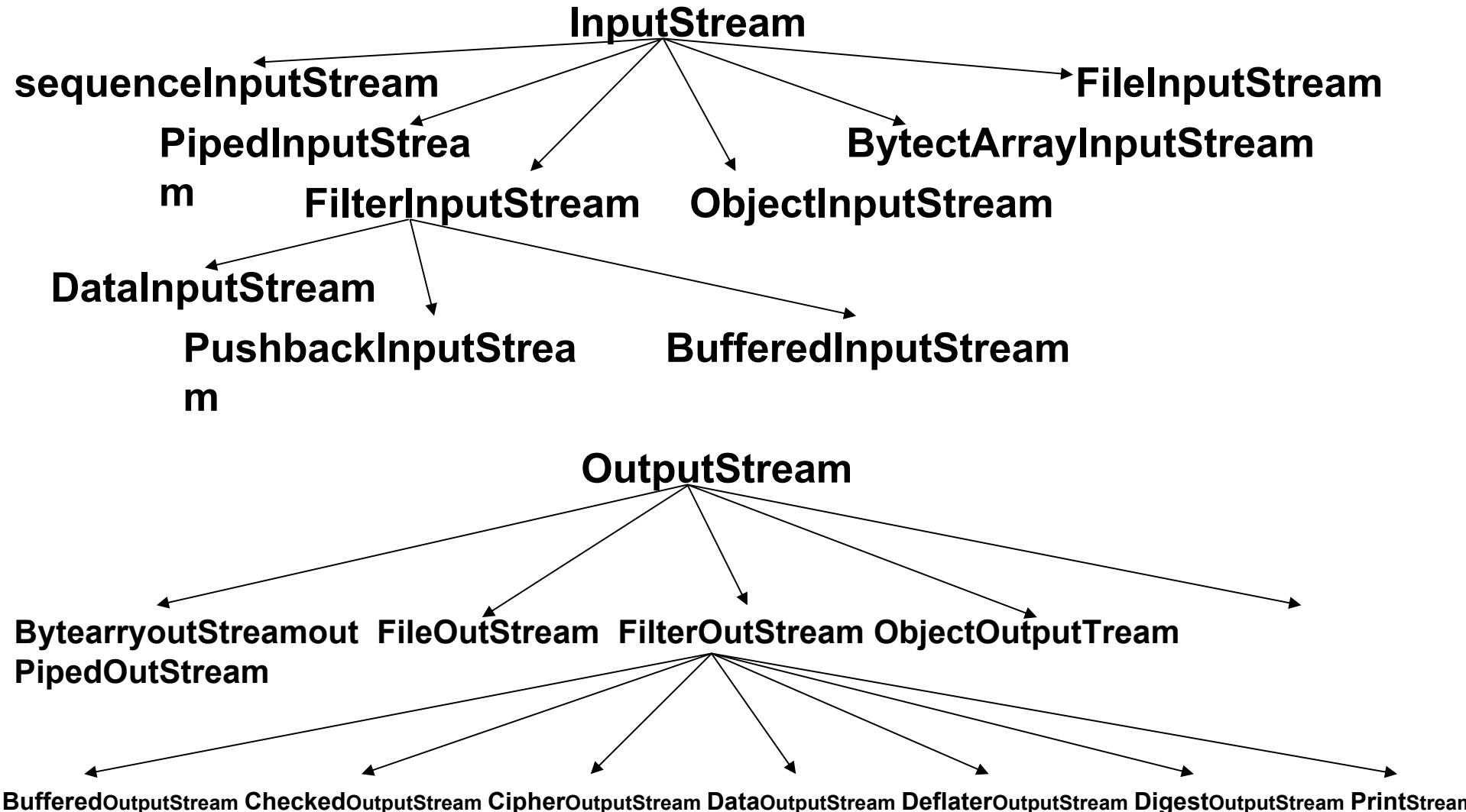
OutputStream Methods

◆ Three **write()** methods

- Ø **void write(int)**
- Ø **void write(byte[])**
- Ø **void write(byte[], int, int)**

◆ Other methods

- Ø **void close()**
- Ø **void flush()**



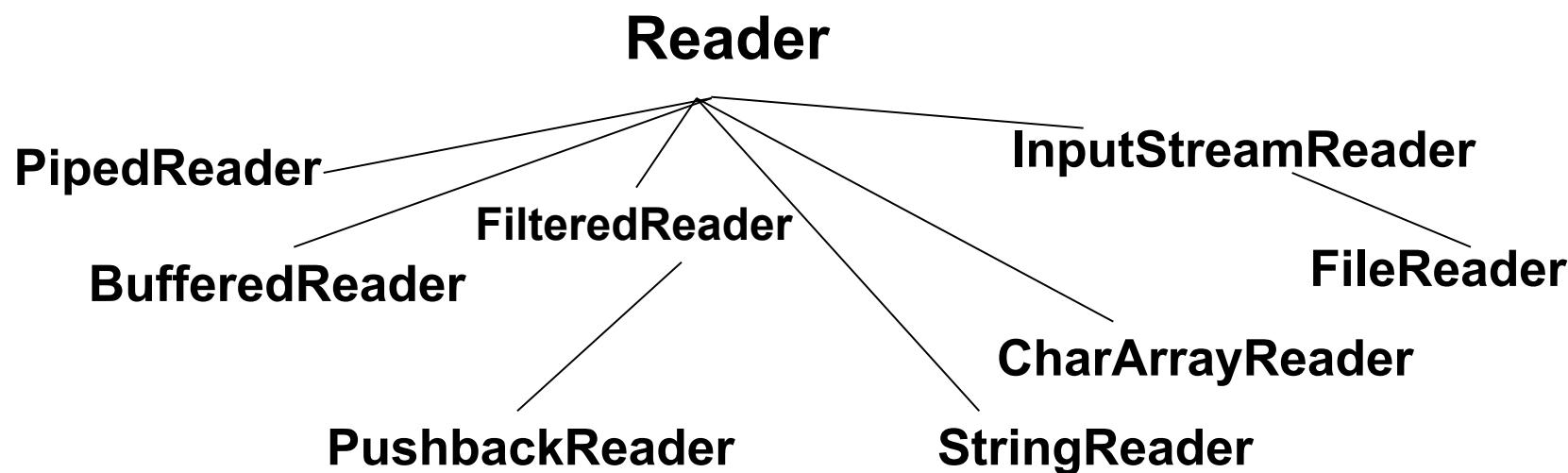
- ◆ Buffering an input/output stream to enhance performance
- ◆ Constructors of the class **BufferedInputStream**
 - ∅ **BufferedInputStream(InputStream in)**
 - ∅ **BufferedInputStream(InputStream in, int size)**
- ◆ Constructors of the class **BufferedOutputStream**
 - ∅ **BufferedOutputStream(OutputStream out)**
 - ∅ **BufferedOutputStream(OutputStream out, int size)**

- ◆ Enabling applications to read/write Java primitive type data
- ◆ Constructors
 - Ø DataInputStream(InputStream in)
 - Ø DataOutputStream(OutputStream out)
- ◆ Methods
 - Ø readBoolean()/writeBoolean
 - Ø readByte()/writeByte()
 - Ø readChar()/writeChar()
 - Ø

PipedInputStream and PipedOutputStream

- ◆ A PipedInputStream connects to a PipedOutputStream
- ◆ One way in which threads provide communication for data exchanging
- ◆ Constructors
 - ∅ PipedInputStream()
 - ∅ PipedInputStream(PipedOutputStream src)
 - ∅ PipedOutputStream()
 - ∅ PipedOutputStream(PipedInputStream sink)

- ◆ The Reader and Writer are abstract classes for reading and writing characters from/to stream
- ◆ Their subclasses must implement *read(char[], int, int)*/ *write(char[], int, int)*, *close()* and *flush()* methods



Encoding String	Description
8859_1	ISO Latin-1
8859_2	ISO Latin-2
8859_3	ISO Latin-3
Cp037	USA, Canada (Bilingual, French), Netherlands, Portugal, Brazil, Australia
Cp1046	IBM Open Edition US EBCDIC
Cp850	PC Latin-1
ISO2022CN	ISO 2022 CN, Chinese
MacRoman	Macintosh Roman
UTF8	Standard UTF-8

1. An InputStreamReader is a bridge from byte stream to character stream
2. An OutputStreamWriter is a bridge from character stream to byte Stream
3. Constructors
 - Ø InputStreamReader(InputStream in, String charsetName)
 - Ø OutputStreamWriter(OutputStream out, String charsetName)

- ◆ Buffering characters so as to provide for the efficient reading/writing characters, chars, and lines
- ◆ Constructors
 - ∅ **BufferedReader(Reader in)**
 - ∅ **BufferedReader(Reader in, int size)**
 - ∅ **BufferedWriter(Writer out)**
 - ∅ **BufferedWriter(Writer out, int size)**
- ◆ Methods
 - ∅ **readLine()**

- ◆ A File object is a representation of either a file or a directory
- ◆ The File class handles all disk operations
- ◆ The File class constructors
 - ∅ File(File parent, String child)
 - ∅ File(String pathname)
 - ∅ File(String parent, String child)
 - ∅ File(URI uri)
- ◆ Directory separators
 - ∅ File.separator s

Methods of The File Class

- ◆ **boolean createNewFile()**
- ◆ **boolean delete()**
- ◆ **boolean mkdir()/mkdirs()**
- ◆ **boolean renameTo(File destination)**
- ◆ **boolean canRead()/canWrite()**
- ◆ **boolean exists()**
- ◆ **String[] list()**
- ◆ **long lastModified()**
- ◆ **String getPath()/getAbsolutePath()**
- ◆ **String getParent()/getName()**

FileNode.java

```
package file;
import java.io.File;
public class FileNode {
    private File f;
    public FileNode(String fName){ f = new File(fName); }
    public FileNode getChild(int index){
        String fPath = f.getAbsolutePath();
        String[] fList = f.list();
        if(fList == null) return null;
        int len = fList.length;
        int j = 0;
        int i = 0;
        for(; i < len; i++){
            File file = new File(fPath + File.separator + fList[i]);
            if(file.isDirectory()){
                if(j == index) break;
                j++;
            }
        }
        return new FileNode(fPath + File.separator + fList[i]);
    }
}
```

FileNode.java(Cont.)

```
public String toString(){ return f.getName();}

public int getCount(){
    String fPath = f.getAbsolutePath();
    String[] fList = f.list();

    if(fList == null) return 0;
    int len = fList.length;
    int j = 0;
    int i = 0;
    for(; i < len; i++){
        File file = new File(fPath + File.separator + fList[i]);
        if(file.isDirectory())
            j++;
    }

    return j;
}
```

FileTreeModel.java

```
package file;
import javax.swing.tree.TreeModel;
import javax.swing.tree.TreePath;
import javax.swing.event.TreeModelListener;

public class FileTreeModel implements TreeModel {
    private String beginPath;

    public FileTreeModel(String beginPath){ this.beginPath = beginPath; }

    public Object getRoot() { return new FileNode(beginPath); }

    public Object getChild(Object parm1, int parm2) {
        FileNode fNode = (FileNode)parm1;
        return fNode.getChild(parm2);
    }

    public int getChildCount(Object parm1) {
        FileNode fNode = (FileNode)parm1;
        return fNode.getCount();
    }
}
```



FileTreeModel.java(cont.)

```
public boolean isLeaf(Object parm1) {  
    FileNode fNode = (FileNode)parm1;  
    return fNode.getCount() == 0;  
}  
  
public void valueForPathChanged(TreePath parm1, Object parm2) {}  
  
public int getChildIndex(Object parm1, Object parm2) {  
    FileNode parent = (FileNode)parm1;  
    FileNode child = (FileNode)parm2;  
    return parent.getIndex(child);  
}  
  
public void addTreeModelListener(TreeModelListener parm1) {  
    listeners.add(parm1);}  
  
public void removeTreeModelListener(TreeModelListener parm1) {  
    listeners.remove(parm1);}  
  
private java.util.Collection listeners = new java.util.Vector();  
}
```

package file;

```
import javax.swing.JFrame;
import javax.swing.JTree;
import javax.swing.JScrollPane;
public class DirBrowser extends JFrame {
    public DirBrowser(String head, String path){
        super(head);
        FileTreeModel model = new FileTreeModel(path);
        JTree tree = new JTree(model);
        JScrollPane pane = new JScrollPane(tree);
        this.getContentPane().add(pane);
        pack();
        setVisible(true);
    }
}
```

DirBrowser.java(cont.)

```
public static void main(String[] args){  
    if(args.length != 1){  
        System.out.println("Usage: java " + DirBrowser.class.getName()  
                           + " <path>");  
        System.exit(1);  
    }  
}
```

```
    DirBrowser dirBrowser1 = new DirBrowser("Browser", args[0]);  
    dirBrowser1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```



FileInputStream and FileOutputStream

◆ The file must exist and can be read when you create a FileInputStream

- ∅ **FileInputStream(File file)**
- ∅ **FileInputStream(String name)**

```
FileInputStream infile =  
    new FileInputStream("myfile.dat");  
FileOutputStream outfile =  
    new FileOutputStream("results.dat");
```

◆ A new file will be created or existed file will be overridden when you create a FileOutputStream

- ∅ **FileOutputStream(File file)**
- ∅ **FileOutputStream(File file, boolean append)**
- ∅ **FileOutputStream(String name)**
- ∅ **FileOutputStream(String name, boolean append)**

- ◆ Convenience for reading/writing character files
- ◆ Constructors
 - ∅ **FileReader(File file)**
 - ∅ **FileReader(String name)**
 - ∅ **FileWriter(File file)**
 - ∅ **FileWriter(String filename)**

DataFileTest.java

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

public class DataFileTest{
    static void writeData(Employee[] e, PrintWriter out)
        throws IOException { out.println(e.length);
        for (int i = 0; i < e.length; i++)
            e[i].writeData(out);
    }

    static Employee[] readData(BufferedReader in) throws IOException{
        int n = Integer.parseInt(in.readLine());
        Employee[] e = new Employee[n];
        for (int i = 0; i < n; i++){
            e[i] = new Employee();
            e[i].readData(in);
        }
        return e;
    }
}
```



DataFileTest.java(cont.)

```
public static void main(String[] args) {  
    Employee[] staff = new Employee[3];  
    staff[0] = new Employee("Harry Hacker", 35500, new Day(1989,10,1));  
    staff[1] = new Employee("Carl Cracker", 75000, new Day(1987,12,15));  
    staff[2] = new Employee("Tony Tester", 38000, new Day(1990,3,15));  
    for (int i = 0; i < staff.length; i++) staff[i].raiseSalary(5.25);  
  
    try{  
        PrintWriter out = new PrintWriter(new FileWriter("employee.dat"));  
        writeData(staff, out); out.close();  
    } catch(IOException e) {  
        System.out.print("Error: " + e); System.exit(1);  
    }  
  
    try { BufferedReader in = new BufferedReader(new  
FileReader("employee.dat"));  
        Employee[] e = readData(in);  
        for (i = 0; i < e.length; i++) e[i].print();  
        in.close();  
    } catch(IOException e) { System.out.print("Error: " + e); System.exit(1); }  
}
```

Employee.java

```
public class Employee {  
    public Employee(String n, double s, Day d) { name = n; salary = s;  
hireDay = d; }  
  
    public Employee() {}  
  
    public void print() { System.out.println(name + " " + salary + " " +  
hireYear()); }  
  
    public void raiseSalary(double byPercent){ salary *= 1 + byPercent /  
100; }  
  
    public int hireYear() { return hireDay.getYear(); }  
  
    public void writeData(PrintWriter out) throws IOException {  
        out.println(name + "|" + salary + "|" + hireDay.getYear() + "|"  
+ hireDay.getMonth() + "|" + hireDay.getDay());  
    }  
}
```



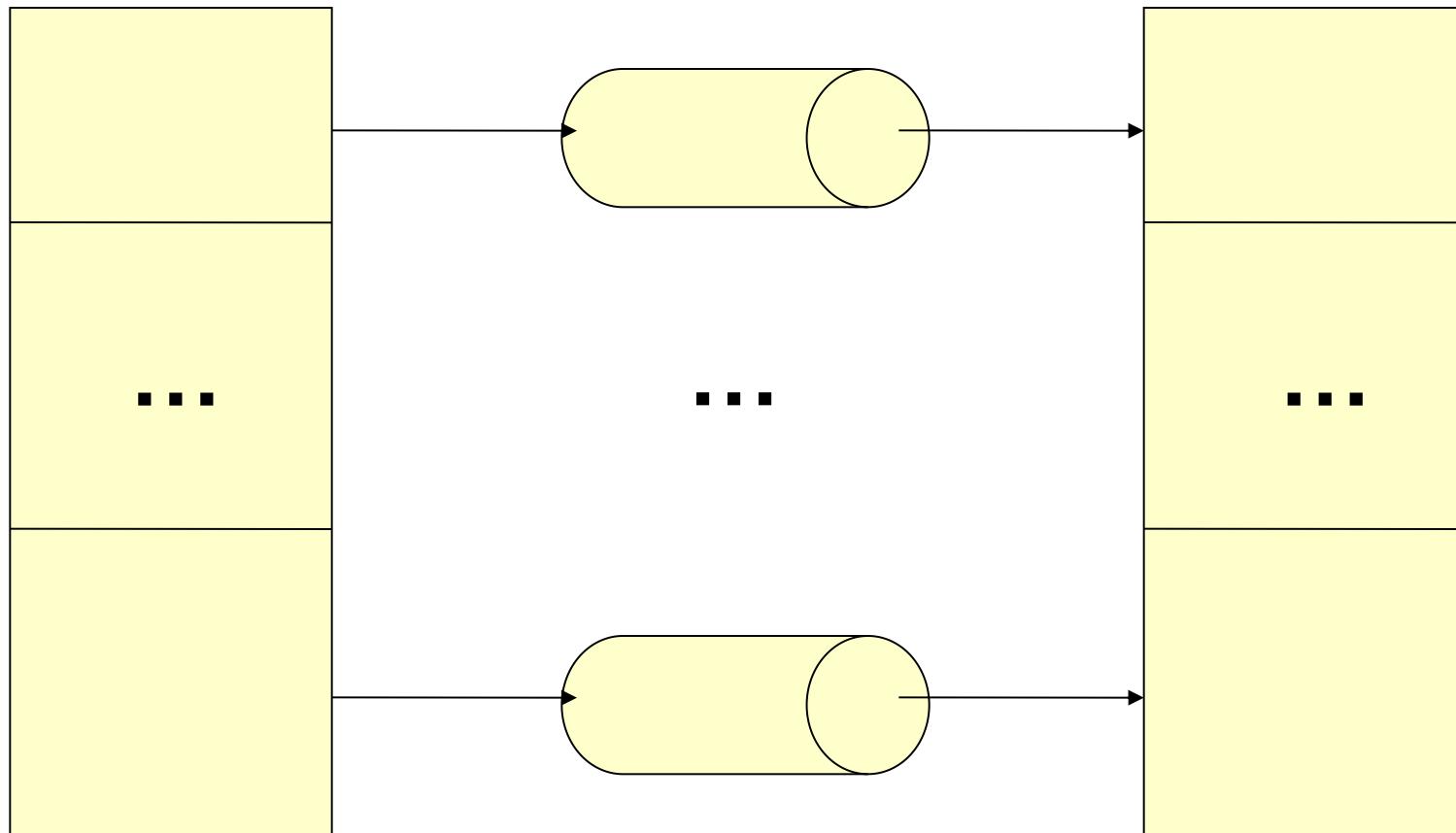
```
public void readData(BufferedReader in) throws  
IOException{  
    String s = in.readLine();  
    StringTokenizer t = new StringTokenizer(s, "|");  
    name = t.nextToken();  
    salary = Double.parseDouble(t.nextToken());  
    int y = Integer.parseInt(t.nextToken());  
    int m = Integer.parseInt(t.nextToken());  
    int d = Integer.parseInt(t.nextToken());  
    hireDay = new Day(y, m, d);  
}  
  
private String name;  
private double salary;  
private Day hireDay;  
}
```

```
public class Day{  
    private int year, month, day;  
  
    public Day(int y, int m, int d ){  
        year = y; month = m; day = d;  
    }  
  
    public int getYear(){ return year;}  
  
    public int getMonth() { return month; }  
  
    public int getDay() { return day; }  
}
```

**Hint: if you want to validate the date, please refer
java.util.Calendar**



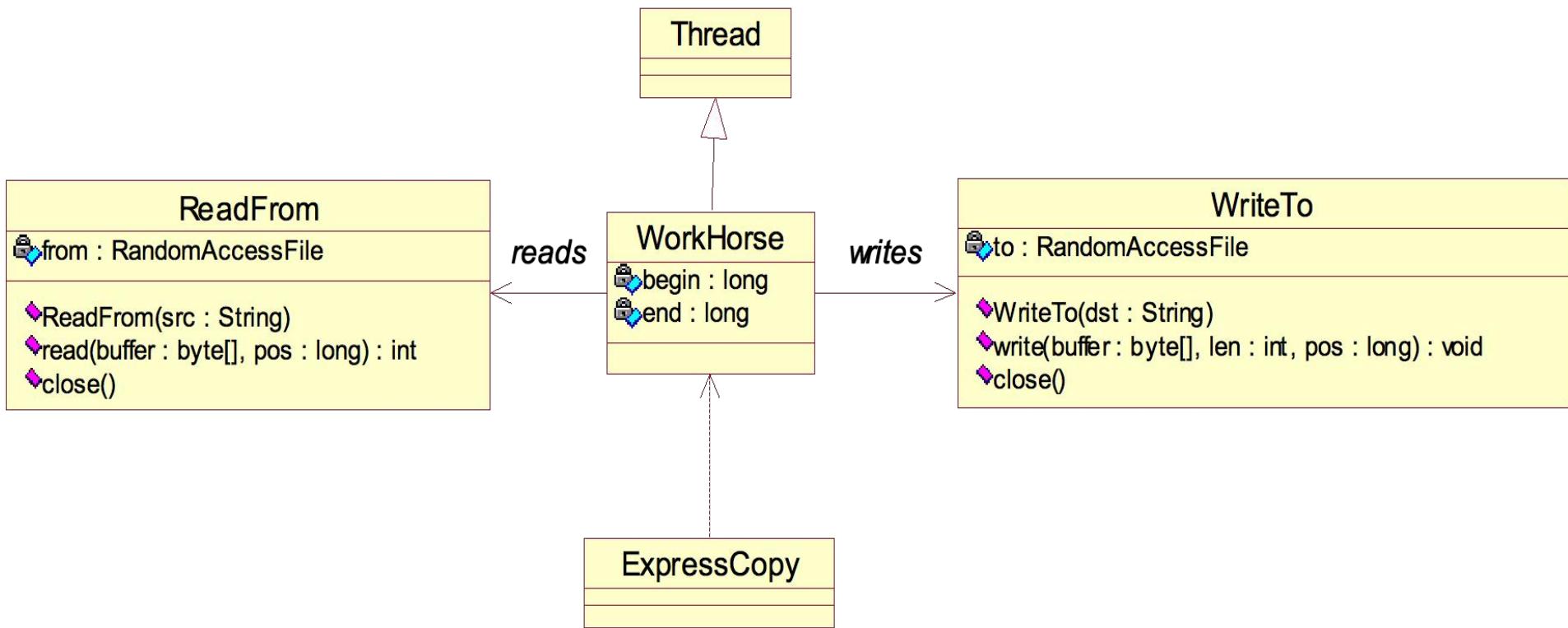
- ◆ **Implementing both DataInput and DataOutput**
- ◆ **Providing capability for reading as well as writing to a file when it is opened**
- ◆ **Providing capability for reading or writing to a specified point within the file by using a file pointer**
- ◆ **Constructors**
 - Ø **RandomAccessFile(File file, String mode)**
 - Ø **RandomAccessFile(String filename, String mode)**
- ◆ **Methods**
 - Ø **read()/write()**
 - Ø **seek(long pointer)**



Source file

Destination file

briup



```
class WorkHorse extends Thread {  
    private long begin, end;  
    private ReadFrom rf;  
    private WriteTo wt;  
  
    public WorkHorse(long begin, long end, ReadFrom rf, WriteTo wt){  
        this.begin = begin;  this.end = end;  this.rf = rf;  this.wt = wt;  
    }  
  
    public void run(){  
        byte[] buffer = new byte[4096];  int len = 0;  
        try {  
            while(begin < end) {  
                len = rf.read(buffer, begin);  
                if(begin + len > end) len = (int)(end - begin);  
                wt.write(buffer, len, begin);  
                begin += len;  
            }  
        }catch(IOException e) { e.printStackTrace(); }  
    }  
}
```

```
class ReadFrom{  
    RandomAccessFile from;  
  
    public ReadFrom(String fromFile) throws IOException{  
        from = new RandomAccessFile(fromFile, "r");  
    }  
  
    synchronized public int read(byte[] buffer, long pos) throws  
IOException{  
    from.seek(pos);  
    return from.read(buffer);  
}  
  
    public void close() throws IOException {  
        from.close();  
    }  
}
```

```
class WriteTo {  
    RandomAccessFile to;  
  
    public WriteTo(String toFile) throws IOException{  
        to = new RandomAccessFile(toFile, "w");  
    }  
  
    synchronized public void write(byte[] buffer, int len, long pos) throws  
IOException{  
    to.seek(pos);  
    to.write(buffer, 0, len);  
}  
  
    public void close() throws IOException {  
        to.close();  
}  
}
```

```
import java.io.*;
public class ExpressCopy {

    public static void main(String[] args) {
        if(args.length != 2) {
            System.out.println("Usage: java " + ExpressCopy.class.getName()
+
                    " sourceFile destFile");
            System.exit(1);
        }

        File f = new File(args[0]);

        if(!f.exists() || f.length() == 0) {
            System.out.println("The source file " + args[0] + " does not exist or
its length is 0");
            System.exit(0);
        }
        long len = f.length();
```

```
ReadFrom rf = null;  WriteTo wt = null;
try {
    rf = new ReadFrom(args[0]);
    wt = new WriteTo(args[1]);
    Thread[] t = new Thread[3];
    for(int i = 0; i < t.length; i++) {
        t[i] = new WorkHorse( i * len / 3 , (i + 1) * len / 3, rf, wt); [i].start();
    }

    for(int i = 0; i < t.length; i++) t[i].join();

}catch(Exception e){ e.printStackTrace();}
finally{
    if(rf != null){ try{ rf.close(); } catch(Exception e){} }
    if(wt != null) { try{ wt.close(); } catch(Exception e) {} }
}
}
```

- ◆ Objects persistence
- ◆ Only the data of an object is serialized
- ◆ Transit data can not be Serialized
- ◆ Objects must implement `java.io.Serializable`

ObjectInputStream and ObjectOutputStream

- ◆ **ObjectOutputStream and ObjectOutputStream can provide an application with persistent storage for graphs of objects**
- ◆ **An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream**
- ◆ **Constructors**

ObjectInputStream(InputStream in)

ObjectOutputStream(OutputStream out)

- ◆ **Methods**

readObject()/writeObject()

◆ In this chapter, we have discussed:

- Ø Navigating a file system
- Ø The java.io hierarchy
- Ø Creating files and file-choosers
- Ø Choosing suitable reader and/or writer
- Ø Operations on files and directories
- Ø Reading, writing and updating data from/to files
- Ø Using serialization mechanism

Formatted I/O & Scanner

u printf is popular with C/C++ developers

Ø Powerful, easy to use

u Finally adding printf to J2SE 5.0 (using varargs)

```
String name="Mashal zhao";
```

```
int l = name.length();
```

```
float w = 8.2f;
```

```
out.printf("-12s is %d long", name, l);
```

```
out.printf("weight = %.2f", w);
```

u Also a simple scanning API: convert text into primitives or Strings

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

```
Int n = s.nextInt();
```



Review Questions

1. (Level 1) What are the main classes for input stream?
2. (Level 1) What are reader/writer classes?
3. (Level 1) What are the bridges between InputStream and Reader?
4. (Level 1) What is object serialization?
5. (Level 2) The File class contains a method that changes the current working directory
 - A.True
 - B.False
6. (Level 2) It is possible to use the File class to list the contents of the current working directory
 - A.True
 - B.False
7. (Level 2) Readers have methods that can read and return floats and doubles
 - A.True
 - B.False

Review Questions – Con’t

8. (Level 3) How many bytes does the following code write to file dest ?

1. try{
2. FileOutputStream fos = new FileOutputStream("dest");
3. DataOutputStream dos = new DataOutputStream(fos);
4. dos.writeInt(3);
5. dos.writeDouble(0.001);
6. dos.close();
7. fos.close();
8. }catch(IOException e){}

- a) A.2
- b) B.8
- c) C.12
- d) D.16
- e) E.It depends on the underlying system

Assignment

- 1. (Level 1) Write a class called BinFileInput which reads binary data from the file created by BinFileOutput.**
- 2. (Level 2) Write a class called FileCopy . The class will get input from one file and copy the content into another file simultaneously(one reader and multiple writers).**
- 3. (Level 3) Write a class called PropertiesFile. It will load data from a file which has “key = value” format. You can get the value by passing the key.**





杰普软件科技有限公司
www.briup.com

公司总部: 上海市闸北区万荣路
1188弄龙软软件园区A栋206室
电话: (021) 56778147
邮政编码: 200436

昆山实训基地: 昆山市巴城学院路
828号昆山浦东软件园北楼4-5-8层
电话: (0512) 50190290-8000
邮政编码: 215311

电邮: training@briup.com
主页: <http://www.briup.com>

Briup High-End IT Training

Module 8

Network

Brighten Your Way And Raise You Up.

Objectives

- ◆ Explain TCP/IP protocols
- ◆ Explain Socket programming
- ◆ Explain ServerSocket programming
- ◆ Explain URL and URLConnection programming
- ◆ Explain UDP Socket programming



Confused Points

- ◆ Choose suitable high level I/O streams for reading/writing different data type information from/to socket to encapsulate raw I/O streams got from socket
- ◆ Use multi-thread to handle concurrency
- ◆ Handle exceptions
- ◆ Understand frequently-used application protocols such as HTTP
- ◆ Write your own application protocol



◆ **java.net**

- ∅ **Socket**
- ∅ **ServerSocket**
- ∅ **DatagramSocket and DatagramPacket**
- ∅ **URL and URLConnection**

◆ **java.io**

- ∅ **InputStream/OutputStream**
- ∅ **BufferedInputStream/BufferedOutputStream**
- ∅ **DataInputStream/DataOutputStream**
- ∅ **BufferedReader/PrintWriter**
- ∅ **ObjectInputStream/ObjectOutputStream**

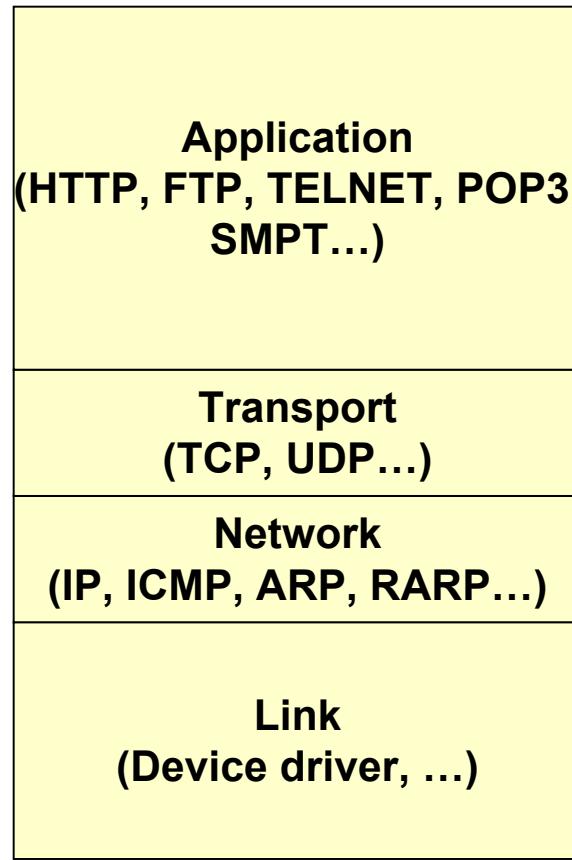
◆ **java.lang**

- ∅ **Thread**
- ∅ **Runnable**

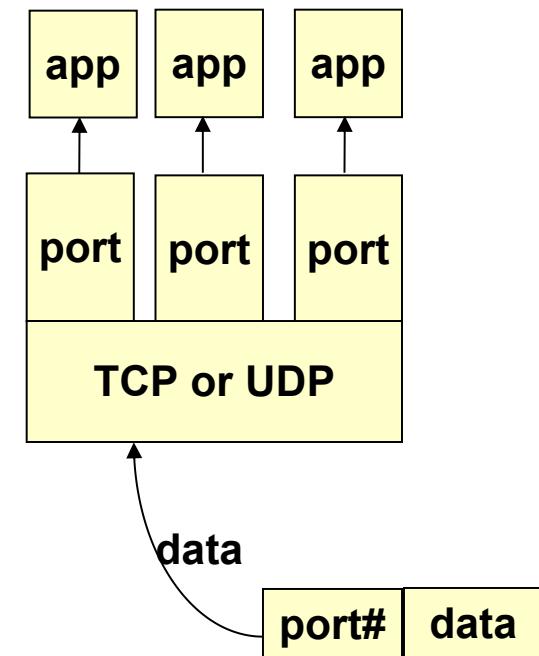
OSI Model



TCP/IP stack



Scenario



briup

- ◆ **Addressing**
- ◆ **Routing**
- ◆ **Flow control**
- ◆ **Connectionless**
- ◆ **Virtual packets**
- ◆ **Unreliable transportation**
- ◆ **Best effort delivery**

- ◆ **Connection orientation**
- ◆ **Full reliable data transportation**
- ◆ **Point to point**
- ◆ **Full duplex communication**
- ◆ **Stream interface**
- ◆ **Reliable connection startup**
- ◆ **Graceful connection shutdown**

◆ Network address

Host Name: bbs.briup.com

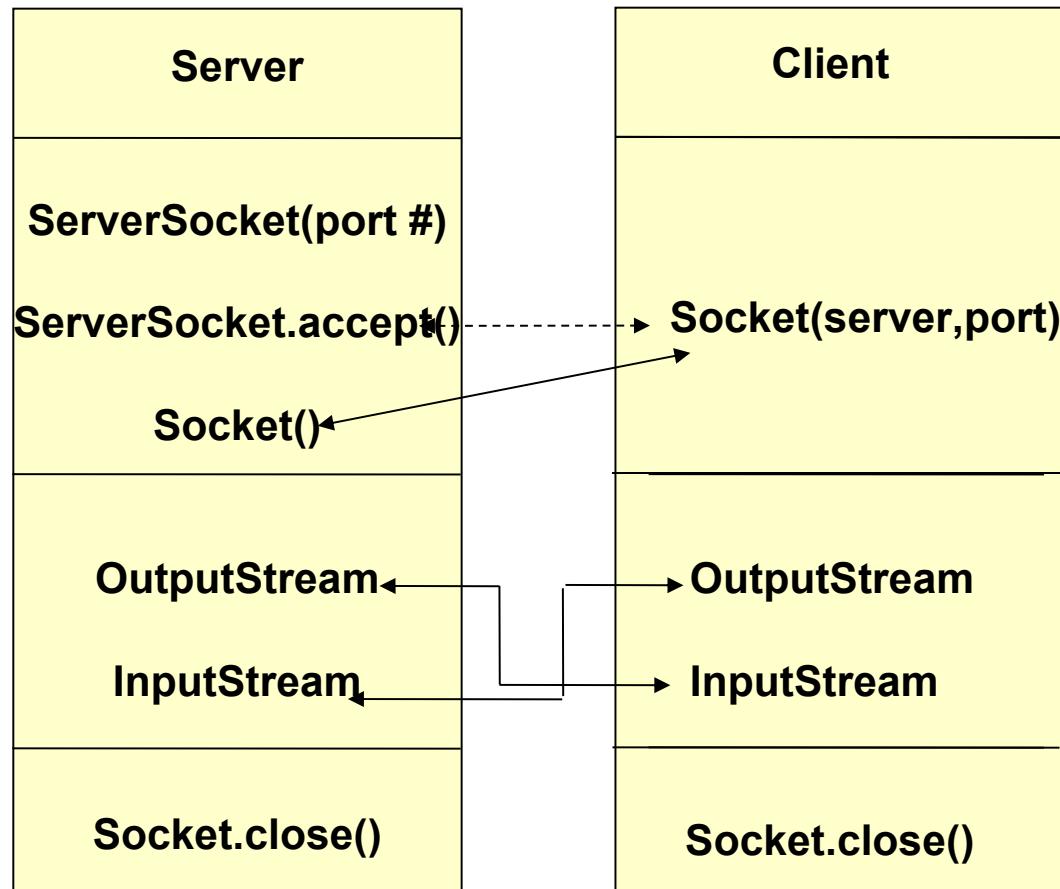
Address: 222.73.0.10

◆ Port number

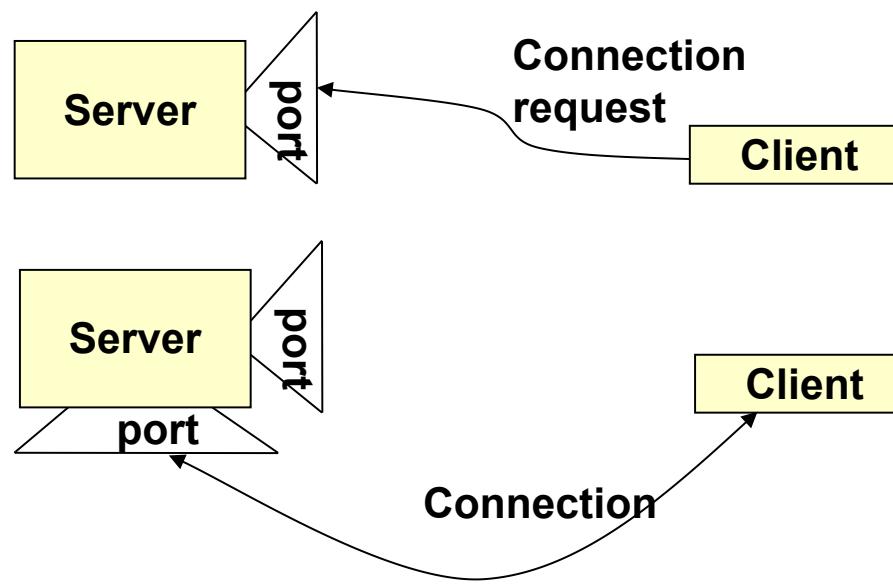
0~65535

◆ Java Networking model

Socket programming

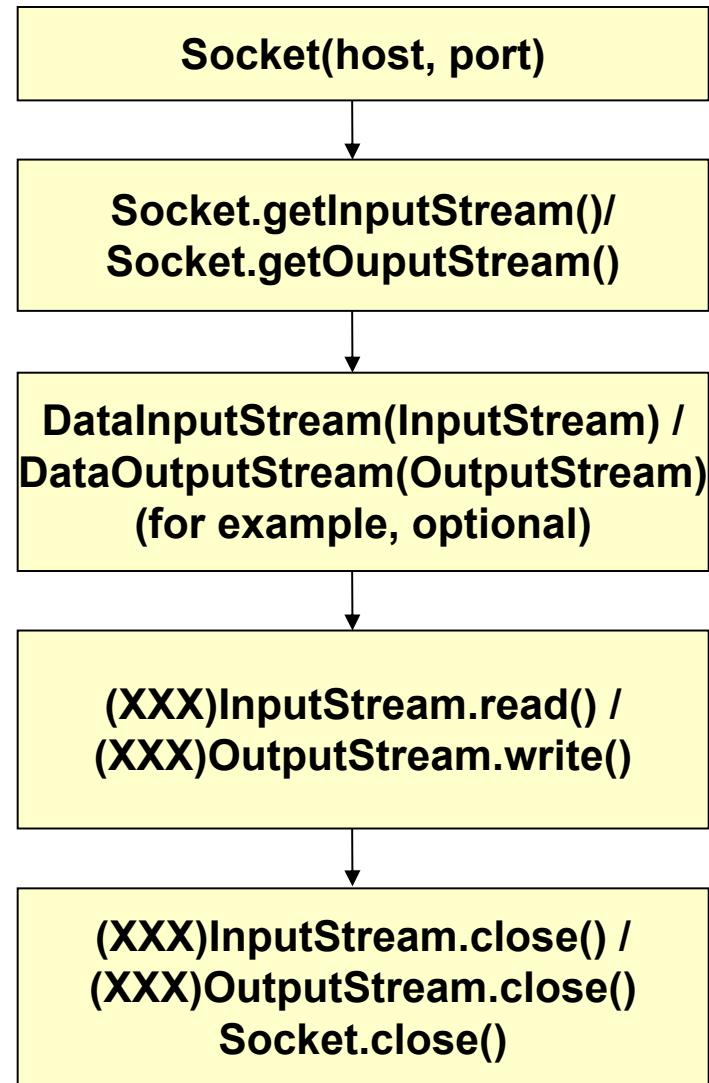


- ◆ A socket is one end-point of a two-way communication link between two programs running on the network
- ◆ A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent



briup

1. Create a socket
2. Get input stream and/or output stream from the connected socket
3. Encapsulate raw input stream /output stream to high level streams according to data type (optional)
4. Receive/send message from/to streams
5. Release resources



- ◆ **Most unix systems provide daytime service based on TCP protocol on port 13. The daytime server accept a client connection request and send back the daytime string followed by a new line character to the client, then close the connection.**
- ◆ **So, we create a socket to connect to the daytime server, encapsulate the input stream got from Socket to a BufferedReader, read one line of string from the BufferedReader and display the string, then release all resources.**

```
//import classes related to socket programming
import java.net.Socket;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

public class TimeClient{
    public static void main(String[] args){
        //The program needs a command line argment to specify the name
        // or IP address of a daytime server
        if(args.length != 1){
            System.out.println("Usage: " + "java " +
                TimeClient.class.getName() + " timeServerName");
            System.exit(1);
        }
        Socket s = null;
        BufferedReader br = null; //Encapsulate the InputStream got from s
        try{
            //step 1. Create a socket
            s = new Socket(args[0], 13);
```

```
//step 2. Get an InputStream from the socket  
InputStream is = s.getInputStream();  
//step 3. Encapsulate the InputStream to a BufferedReader  
br = new BufferedReader( new InputStreamReader(is));  
//step 4. Read the daytime string from the reader  
String str = br.readLine();  
System.out.println(str);  
} catch(IOException e){ //handle exception  
e.printStackTrace();  
}finally{  
//step 5. Release resources  
if(br != null) try{ br.close(); }catch(IOException e){}  
if(s != null) try{ s.close(); }catch(IOException e){}  
}//end finally  
}//end main  
}//end class
```

Socket

<<constructor>>

+Socket()

+Socket(host: String, port: int)

+Socket(host: String, port: int, localAddr: InetAddress, localPort : int)

+getInputStream() : InputStream

+getOutputStream() : OutputStream

+close(): void

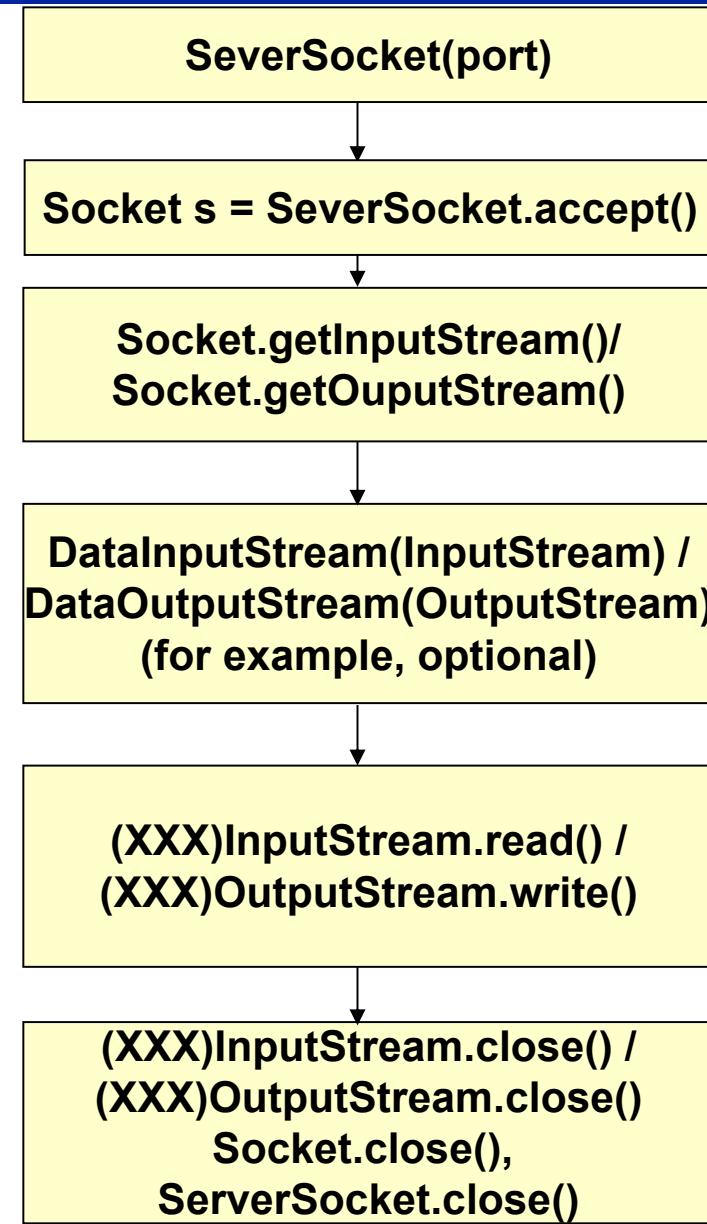
+setSoTimeout(timeout : int) : void

+setKeepAlive(on: boolean) : void

+setSoLinger(on: boolean, linger: int) void



1. Create a server socket
2. Accept a connection request from a client and return a new socket
3. Get input stream / output stream from the accepted socket
4. Encapsulate raw input stream /output stream to high level streams according to data type (optional)
5. Receive/send message from/to streams
6. Release resources



A ServerSocket Sample—Daytime Server

- ◆ The daytime server based on TCP listens on port 13, it accepts connection request from a client and generate a new socket.
- ◆ Get a OutputStream from the new socket and encapsulate it to a PrintWriter
- ◆ Prepare the daytime string and write it to the PrintWriter
- ◆ Close resources including PrintWriter, Socket and ServerSocket



A ServerSocket Sample—Daytime Server(Cont.)

```
//import classes related to server socket programming
import java.net.ServerSocket
import java.net.Socket;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.Calendar;
import java.util.Date;
public class DaytimeServer{
    public static void main(String[] args){
        ServerSocket ss = null;
        Socket s = null; //store the return from ss.accept()
        PrintWriter pw = null; //encapsulate the OutputStream got from
        socket
        try{
            //step 1. Create a listen socket
            ss = new ServerSocket(13);
            //step 2. Accept connection request from a client
            s = ss.accept();
            //step 3. Get a OutputStream from the socket
            OutputStream os = s.getOutputStream();
            //step 4. Encapsulate os to a PrintWriter
            pw = new PrintWriter(os);
            pw.println(new Date().toString());
            pw.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



```
        pw = new PrintWriter(os);
//prepare daytime string
Calendar cal = Calendar.getInstance();
Date d = new Date(cal.getTimeInMillis());
String dtStr = d.toString();
//step 5. Write daytime string to the client
pw.println(dtStr);
}catch(IOException e){ //handle exception
    e.printStackTrace();
}finally{
//step 6. Release resources
    if(pw != null) pw.close();
    if(s != null) try{ s.close(); }catch(IOException e){}
    if(ss != null) try{ ss.close(); } catch(IOException e){}
} //end finally
}//end main
}//end class
```



ServerSocket

<<constructor>>

+ServerSocket(port : int)

+ServerSocket(port : int, backlog : int)

+accept() : Socket

+close(): void

+setReuseAddress(on : boolean)

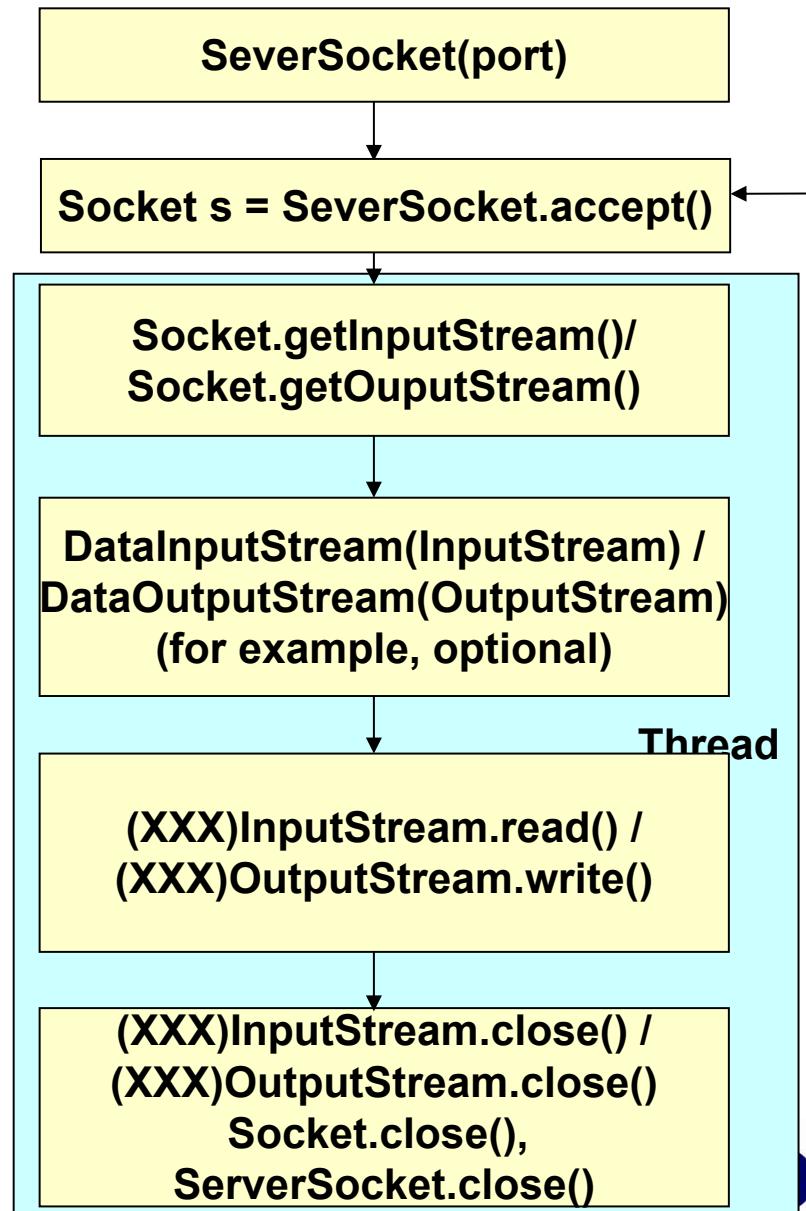
+setSoTimeout(timeout : int) : void



Steps to Handle Concurrency in Socket Server Programming

- ◆ Multiple clients access a server at the same time, each client needs a connection, so one thread per connection

- 1. Create a server socket**
- 2. Accept a connection request from a client and return a new socket, then generate a service thread (repeat)**
- 3. In thread get input stream / output stream from the accepted socket**
- 4. In thread encapsulate raw input stream /output stream to high level streams according to data type (optional)**
- 5. In thrad receive/send message from/to streams**
- 6. In thread Release resources**



- ◆ The daytime server based on TCP listens on port 13, it accepts connection request from a client and generate a new socket.
- ◆ Fork a new thread and pass the new socket to the thread
- ◆ In the thread get a OutputStream from the new socket and encapsulate it to a PrintWriter
- ◆ In the thread prepare the daytime string and write it to the PrintWriter
- ◆ In the thread close resources including PrintWriter, Socket



```
//import classes related to server socket programming
import java.net.ServerSocket
import java.net.Socket;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.IOException;
import java.util.Calendar;
import java.util.Date;
public class DaytimeServer extends Thread{
    public static void main(String[] args){
        ServerSocket ss = null;
        Socket s = null; //store the return from ss.accept()
        //PrintWriter pw = null; //encapsulate the OutputStream got from
        socket
        try{
            //step 1. Create a listen socket
            ss = new ServerSocket(13);
            //step 2. Accept connection request from a client
            while(true){
                s = ss.accept();
                new DaytimeServer(s).start();
            }
        }/end while
    }
}
```



```
 }catch(IOException e){ //handle exception  
     e.printStackTrace();  
 }finally{  
     //step 6. Release resources  
     //if(pw != null) pw.close();  
     //if(s != null) try{ s.close(); }catch(IOException e){}  
     if(ss != null) try{ ss.close(); } catch(IOException e){}  
 } //end finally  
}//end main
```

```
public DaytimeServer(Socket s){  
    this.s = s;  
} public void run(){  
    try{ //step 3. Get a OutputStream from the socket  
        OutputStream os = s.getOutputStream();  
        //step 4. Encapsulate os to a PrintWriter  
        PrintWriter pw = new PrintWriter(os);
```



```
//prepare daytime string
Calendar cal = Calendar.getInstance();
Date d = new Date(cal.getTimeInMillis());
String dtStr = d.toString();
//step 5. Write daytime string to the client
pw.println(dtStr);
}catch(IOException e){ e.printStackTrace(); }
finally{ s.close(); } //step 6. Release resource
}
private Socket s;
}//end class
```



- ◆ The URL and URLConnection classes encapsulate much of the complexity of retrieving information from a remote site
- ◆ Supporting both HTTP and FTP resources
 - ∅ URL url = new URL("http://www.sina.com.cn")
 - ∅ URL url = new URL("ftp://username:password@192.168.1.200")
- ◆ Opening a URL connection
 - ∅ URLConnection urlconn = url.openConnection()
- ◆ Just opening an input stream
 - ∅ InputStream in = url.openStream()

- 1. Create a URL**
- 2. Just reading, call URL.openStream to get an InputStream and skip 3;
reading and writing, call URL.openConnection to generate a
URLConnection, then call URLConnection.connect to establish an
connection to the server**
- 3. Get input stream / output stream from the URLConnection**
- 4. Encapsulate raw input stream /output stream to high level streams
according to data type (optional)**
- 5. Receive/send message from/to streams**

An URL Example

```

package URLConnectionTest;
import java.io.*;
import java.net.*;
import java.util.*;

public class URLConnectionTest
{
    public static void main(String[] args)
    {
        try
        {
            String urlName;
            if (args.length > 0)
                urlName = args[0];
            else
                urlName = "http://java.sun.com";
            //step 1. Create a URL
            URL url = new URL(urlName);
            URLConnection connection = url.openConnection();
            //step 2. Connect to the server
            connection.connect();

            // print header fields

            int n = 1;
            String key;
            while ((key = connection.getHeaderFieldKey(n)) != null)
            {
                String value = connection.getHeaderField(n);
                System.out.println(key + ": " + value);
                n++;
            }

            // print convenience functions
        }
        System.out.println("-----");
        System.out.println("getContentType: "
            + connection.getContentType());
        System.out.println("getContentLength: "
            + connection.getContentLength());
        System.out.println("getContentEncoding: "
            + connection.getContentEncoding());
        System.out.println("getDate: "
            + connection.getDate());
        System.out.println("getExpiration: "
            + connection.getExpiration());
        System.out.println("getLastModified: "
            + connection.getLastModified());
        System.out.println("-----");

        //step 3 and 4. Get an InputStream and encapsulate it
        BufferedReader in = new BufferedReader(new
            InputStreamReader(connection.getInputStream()));

        // print first ten lines of contents

        String line;
        n = 1;
        //step 5. Read info from the stream
        while ((line = in.readLine()) != null && n <= 10)
        {
            System.out.println(line);
            n++;
        }
        if (line != null) System.out.println("...");

        catch (IOException exception)
        {
            exception.printStackTrace();
        }
    }
}

```

- ◆ Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web

- ◆ The format of an URL

communication protocol://hostname:port/file

- ◆ Constructors

- Ø URL(String spec)

- Ø URL(String protocol, String host, int port, String file)

- Ø URL(URL context, String file)

- ◆ Methods

- Ø openStream()

- Ø openConnection()

- ◆ The superclass of all classes that represent a communications link between the application and a URL
- ◆ Constructors
 - ∅ `URLConnection(URL)`
- ◆ Methods
 - ∅ `connect()`
 - ∅ `getInputStream()`
 - ∅ `getOutputStream()`

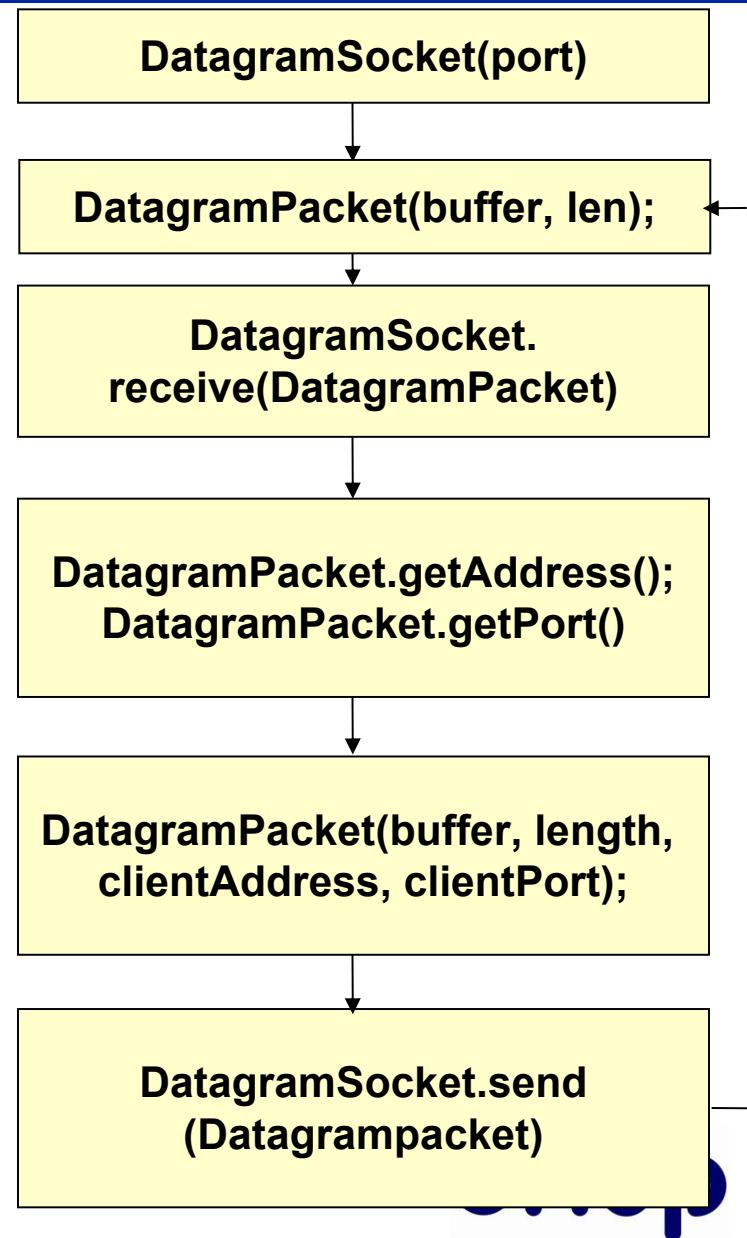
- ◆ **Connectionless protocol**
- ◆ **Not reliable transfer**
- ◆ **DatagramSocket and DatagramPacket support Java UDP**

- ◆ **Implement a connectionless packet delivery service**
- ◆ **Each message is routed from one machine to another based solely on information contained within that packet**
- ◆ **Constructors**
 - ∅ **DatagramPacket(byte[] buf, int length): Constructs a DatagramPacket for receiving packets of length length**
 - ∅ **DatagramPacket(byte[] buf, int length, InetAddress address, int port): Constructs a datagram packet for sending packets of length length to the specified port number on the specified host**
- ◆ **Methods**
 - ∅ **getAddress()/getSockAddress()/getPort()**
 - ∅ **setAddress()/setSockAddress()/setPort()**

- ◆ A datagram socket is the sending or receiving point for a packet delivery service
- ◆ Constructors
 - Ø DatagramSocket()
 - Ø DatagramSocket(int port)
 - Ø DatagramSocket(int port, InetAddress address)
- ◆ Methods
 - Ø connect()/close()
 - Ø receive()/send()

Typical Steps to Write a UDP Server

1. Create a datagram socket bound to a port
2. Prepare a datagram packet to receive message from clients
3. Receive message on datagram socket and put the message into datagram packet
4. Get the info of the client protocol address
5. Prepare a datagram packet which includes message sent back to the client
6. Send datagram packet to the client
7. Repeat 2 - 6



A UDP Server Sample—Daytime Server

- ◆ The daytime server based on UDP bound to port 13, it must receive message from a client first because it has no sense of the protocol address info of a client
- ◆ So, a client can send any message to the server because the server ignore it
- ◆ The server get address info from the inbound datagram packet
- ◆ Prepare the daytime string and put it in the outbound datagram packet
- ◆ Send the outbound datagram packet from the datagram socket to the client

The logo consists of the word "briup" in a lowercase, bold, sans-serif font. The letters are a dark blue color.

A UDP Server Sample—Daytime Server(cont.)

```

package sample;
import java.io.*;
import java.net.*;
import java.util.*;

public class UDPDaytimeServer{
//This method retrieves the current time on the server
    public byte[] getTime(){
        Date d= new Date();
        return d.toString().getBytes();
    }
// Main server loop.
    public void go() throws IOException {
        DatagramSocket datagramSocket;
        DatagramPacket inDataPacket;
            // Datagram packet from the client
        DatagramPacket outDataPacket;
            // Datagram packet to the client
        InetAddress clientAddress; // Client return address
        int clientPort; // Client return port
        byte[] msg= new byte[10]; // Incoming data buffer. Ignored.
        byte[] time; // Stores retrieved time

        // Allocate a socket to man port 13 for requests.
        datagramSocket = new DatagramSocket(13);
        System.out.println("UDP server active on port 13");

        // Loop forever
        while(true) {
            // Set up receiver packet. Data will be ignored.
            inDataPacket = new DatagramPacket(msg, msg.length);
            // Get the message.
            datagramSocket.receive(inDataPacket);
            // Retrieve return address information, including InetAddress
            // and port from the datagram packet just received.
            clientAddress = inDataPacket.getAddress();
            clientPort = inDataPacket.getPort();
    }
}

```

```

// Get the current time.
    time = getTime();

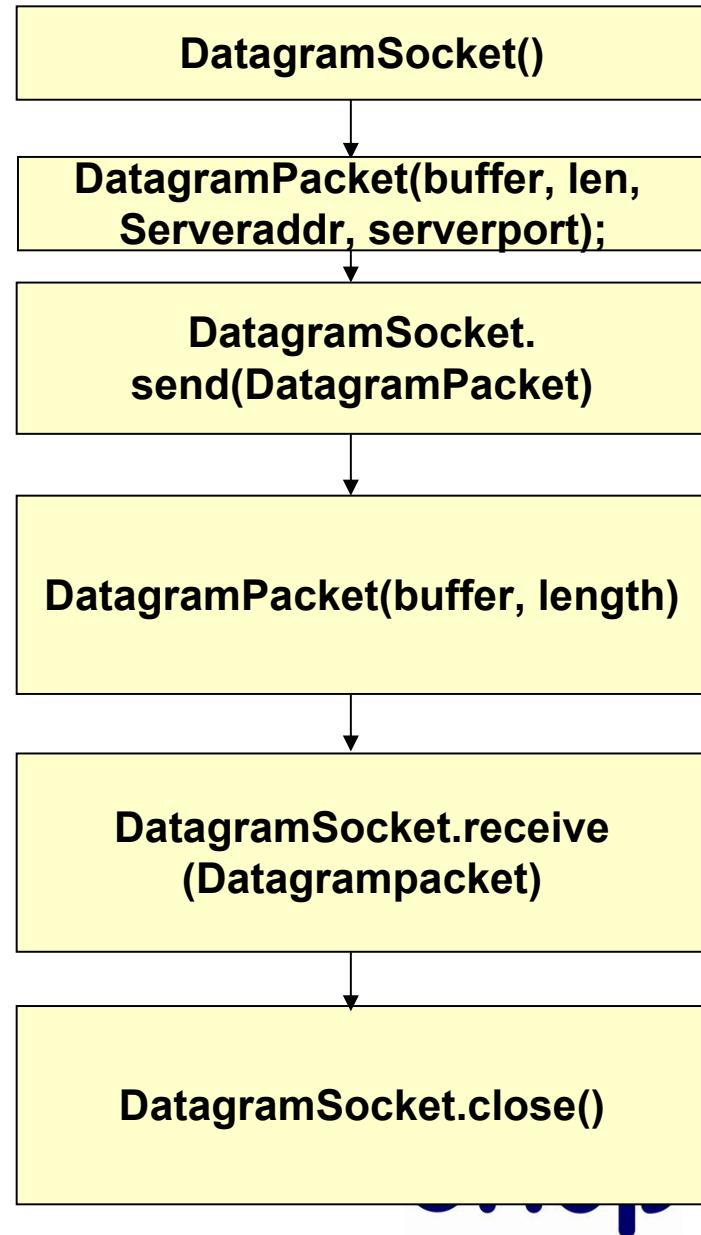
    //set up a datagram to be sent to the client using the
    //current time, the client address and port
    outDataPacket = new DatagramPacket
        (time, time.length, clientAddress, clientPort);

    //finally send the packet
    datagramSocket.send(outDataPacket);
}

public static void main(String args[]) {
    UdpDaytimeServer udpServer =
        new UdpDaytimeServer();
    try {
        udpServer.go();
    } catch (IOException e) {
        System.out.println
            ("IOException occured with socket.");
        System.out.println (e);
        System.exit(1);
    }
}

```

- 1. Create a datagram socket**
- 2. Prepare a datagram packet which includes message and the server protocol address info**
- 3. Send the datagram packet to the server from the datagram socket**
- 4. Prepare a datagram socket which includes a buffer and its length to receive response from the server**
- 5. Receive message from the datagram socket and put the message into the datagram packet prepared in the last step**
- 6. Close the datagram socket**



- ◆ The daytime client based on UDP get the daytime string from a udp daytime server bound to port 13
- ◆ First the client prepare a outbound datagram packet which includes the server protocol address info and a buffer
- ◆ Send the outbound datagram packet to the server
- ◆ Prepare an inbound datagram packet in which there is a buffer to holds daytime string sent back from the server
- ◆ Receive message from the datagram socket and put the message into the inboud datagram packet

A UDP Client Sample—Daytime Client(cont.)

```

package sample;
import java.io.*;
import java.net.*;

public class UDPClient {

    public void go() throws IOException,
                           UnknownHostException {
        DatagramSocket datagramSocket;
        DatagramPacket outDataPacket;
            // Datagram packet to the server
        DatagramPacket inDataPacket;
            // Datagram packet from the server
        InetAddress serverAddress; // Server host address
        byte[] msg = new byte[100]; // Buffer space.
        String receivedMsg;
            // Received message in String form.
        // Allocate a socket by which messages are
                           sent and received.
        datagramSocket = new DatagramSocket();

        // Server is running on this same machine
                           for this example.
        // This method can throw an UnknownHostException.
        serverAddress = InetAddress.getLocalHost();

        // Set up a datagram request to be sent to the server.
        // Send to port 13
        outDataPacket = new DatagramPacket(msg,
                                           1, serverAddress, 13);
        // Make the request to the server.
        datagramSocket.send(outDataPacket);
        // Set up a datagram packet to receive
                           server's response.
        inDataPacket = new DatagramPacket
                      (msg, msg.length);
    }
}

```

```

// Receive the time data from the server
datagramSocket.receive(inDataPacket);

// Print the data received from the server
receivedMsg = new String
              (inDataPacket.getData(), 0, inDataPacket.
               getLength());
System.out.println(receivedMsg);

//close the socket
datagramSocket.close();
}

public static void main(String args[]) {
    UDPClient udpClient = new UDPClient();
    try {
        udpClient.go();
    } catch (Exception e) {
        System.out.println
                  ("Exception occured with socket.");
        System.out.println (e);
        System.exit(1);
    }
}

```

◆ In this chapter, we have discussed:

- u Creating network connection
- u Understanding TCP/IP and UDP protocols
- u Using ServerSocket and Socket to build server and client applications
- u Using DatagramPacket and DatagramSocket to create UDP connections

Review Questions

- 1. (Level 1)What is the difference between UDP and TCP?**
- 2. (Level 1)What is an URL connection?**
- 3. (Level 1)What are the two main classes for TCP networking?**
- 4. (Level 1)What are the two main classes for UDP networking?**

- 1. (Level 2) Write two classes called TcpClient and TcpServer . The TcpClient Will connect to TcpServer using TCP Socket/ServerSocket and send multiple lines.The TcpServer will receive those lines and print them out to the screen.**

- 2. (Level 3) Write two classes called ChatClient and ChatServer .The ChatServer will modify the TcpServer to be multi-threaded and be able to response to multiple clients simultaneously.**



briup