

Lab 5

- Bubble sort algorithm- 1 point
- Binary search algorithm- 2 point
- Letter and character count- 1 point
- Score:
 - Menu- 2 point
 - Interface- 1 point
 - Abstract- 1 point
 - Inheritance- 1 point
 - Static function1 point

Lab 6

- Build a class that describes the Cow and its actions: walking, eating, make sound, ...
- Build a class that describes the Bird and its actions: walking, eating, make sound, ...
 - S1: Implement 2 classes Cow and Bird.
 - S2: Create an Animal class that has common methods. Then the Cow and Dog class will inherit from the Animal class, then the Cow and Bird class will override the methods in its way.
 - S3: Create an abstract class Animal that has properties and methods. Class Cow, Bird extends from Animal.
- One more feature for the bird is its ability to fly
 - S4: Create an abstract Animal with properties and methods. Class Cow, Bird extends from Animal. Bird has a Fly method.
 - S5: Create an abstract Animal with properties and methods. Class Cowl, Bird extends from Animal. Create an interface FlyAbility and Bird will implement FlyAbility interface.

Labs

- Interface - “ a kind of”+ multi Inheritance
- Abstract Class - “is a”+ single Inheritance

Introduction

- A class can contain no data field or some data fields
- Some operations on numbers are critical such as converting a string to number, ...
- Java libraries have classes which wrap a number (primitive type) in it and support operations on numbers. They are called as wrapper classes.
- String is a common data type and it is a pre-defined class in Java library.
- The **java.lang** package contains all of them

Numbers Classes

- Java platform provides *wrapper* classes for each of the primitive data types.

- `java.lang.Object`
 - `java.lang.Boolean` (implements `java.lang.Comparable<T>`, `java.io.Serializable`)
 - `java.lang.Character` (implements `java.lang.Comparable<T>`, `java.io.Serializable`)
 - `java.lang.Character.Subset`
 - `java.lang.Character.UnicodeBlock`
 - `java.lang.Math`
 - `java.lang.Number` (implements `java.io.Serializable`)
 - `java.lang.Byte` (implements `java.lang.Comparable<T>`)
 - `java.lang.Double` (implements `java.lang.Comparable<T>`)
 - `java.lang.Float` (implements `java.lang.Comparable<T>`)
 - `java.lang.Integer` (implements `java.lang.Comparable<T>`)
 - `java.lang.Long` (implements `java.lang.Comparable<T>`)
 - `java.lang.Short` (implements `java.lang.Comparable<T>`)

All of the numeric wrapper classes are subclasses of the abstract class `Number`.

Numbers Classes: A Declaration

public final class Integer extends Number implements Comparable<Integer>

Fields

Modifier and Type	Field and Description
static int	BYTES The number of bytes used to represent a int value in two's complement binary form.
static int	MAX_VALUE A constant holding the maximum value an int can have, $2^{31}-1$.
static int	MIN_VALUE A constant holding the minimum value an int can have, -2^{31} .
static int	SIZE The number of bits used to represent an int value in two's complement binary form.
static Class<Integer>	TYPE The Class instance representing the primitive type int.

Numbers Classes: A Declaration

public final class Integer extends Number implements Comparable<Integer>

We can not create a sub-class of a wrapper class

Constructors

Constructor and Description

Integer(int value)

Constructs a newly allocated Integer object that represents the specified int value.

Integer(String s)

Constructs a newly allocated Integer object that represents the int value indicated by the String parameter.

Numbers Classes: A Declaration

public final class Integer extends **Number** implements **Comparable<Integer>**

Some
common
methods

Wrapper
classes are
immutable
(non-
changeable)
because they
do not have
setters

```
byte, short, ...    byteValue(), shortValue(), intValue(), longValue(),
                    floatValue(), doubleValue()

static int          compare(int x, int y)

int                compareTo(Integer anotherInt)

static int          compareUnsigned(int x, int y)

boolean            equals(Object obj)

static Integer      getInteger(String nm), getInteger(String nm, int val),
                    getInteger(String nm, Integer val)

static int          lowestOneBit(int i)

static int          max(int a, int b), min(int a, int b)

static int          parseInt(String s), parseInt(String s, int radix)
                    parseUnsignedInt(String s), parseUnsignedInt(String s, int radix)

static String       toBinaryString(int i), toHexString(int i), toOctalString(int i)
                    toString(int i), toString(int i, int radix), toUnsignedString(int i)

String             toString()

static long         toUnsignedLong(int x)

static String       toUnsignedString(int i, int radix)

static Integer      valueOf(int i), valueOf(String s), valueOf(String s, int radix)
```

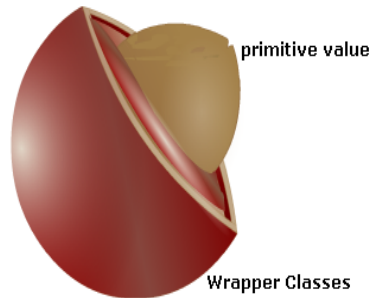

Numbers Classes...

- We use a Number object rather than a primitive when:
 - As an argument of a method that expects an object.
 - To use constants defined by the class, such as `MIN_VALUE` and `MAX_VALUE`.
 - To use class methods for `converting` values to and from other primitive types.

Numbers Classes- A Demo

java.lang.Object

- java.lang.Boolean
- java.lang.Character
- java.lang.Number
 - java.lang.Byte
 - java.lang.Double
 - java.lang.Float
 - java.lang.Integer
 - java.lang.Long
 - java.lang.Short



`int intValue()`

`Integer(int value)`
`Integer(String s)`

5

obj
x 5

`int x= 5;`
`Integer obj = new Integer(5);`
`Integer obj2 = new Integer ("123");`

```

1 public class WrapperDemo {
2     public static void main(String[] args) {
3         int n=7;
4         Integer intObj=new Integer (5);
5         System.out.println(intObj);
6         System.out.println(intObj.toString());
7         intObj=n; // boxing
8         System.out.println(intObj);
9         int y= intObj * 2; // unboxing
10        int z= intObj.intValue();
11        System.out.println("y= " + y + ", z= " + z);
12        n= Integer.parseInt("1234");
13        System.out.println("n= " + n);
14        n= Integer.parseInt("1A", 16);
15        System.out.println("n= " + n);
16    }
17 }
    
```

Output - Chapter08 (run)

```

5
5
7
y= 14, z= 7
n= 1234
n= 26
    
```

Boxing/auto boxing:
encapsulating/wrapping a
primitive value to an object.
Unboxing: get primitive value
wrapped in a wrapper object.

The `java.lang.Math` class

- The `Math` class in the `java.lang` package provides methods and constants for doing more advanced mathematical computation, including:
 - **Constants and Basic Methods:** `Math.E`, `Math.PI`,...
 - Basic static methods: `ceil(double d)`, `floor(double d)`, `abs(int i)`...
 - **Exponential and Logarithmic Methods:** `exp(double d)`, `sqrt(double d)`, `pow(double base, double exponent)`
 - **Trigonometric Methods:** `cos(double d)`, `sin(double d)`
 - **Random Numbers:** The `random()` method returns a pseudo-randomly selected number between 0.0 and 1.0.

Auto boxing and Unboxing (1)

- Java 5.0 introduces two very simple but convenient functions that unwrap wrapper objects and wrap up primitives.
- Converting a primitive value into an object of the corresponding wrapper class is called auto boxing.
- Converting an object of a wrapper type to its corresponding primitive value is called unboxing.

Auto boxing and Unboxing...

- Sample of auto boxing and unboxing

Integer wrappedInt = 25; //boxing or auto boxing

```
Double area(double radius) {  
    return Math.PI * radius * radius; //boxing  
}
```

```
Integer wi = 234;  
int times9 = wi * 9; //unboxing
```

Characters

- Unicode character, 2 bytes
- Character class also offers a number of useful class (i.e., static) methods for manipulating characters.
- `Character ch = new Character('a');`
- Some methods in this class
 - `boolean isLetter(char ch)/ isDigit(char ch)/ isUpperCase(char ch)`
 - `char toUpperCase(char ch) ...`
- A character preceded by a backslash (\) is an *escape sequence* and has special meaning to the compiler.

Strings

- Java uses the **String**, **StringBuffer**, and **StringBuilder** classes to encapsulate strings of characters (16-bit Unicode).

java.lang.Object

java.lang.String (implements java.lang.CharSequence,
java.lang.Comparable<T>, java.io.Serializable)

java.lang.StringBuffer (implements java.lang.CharSequence,
java.io.Serializable)

java.lang.StringBuilder (implements
java.lang.CharSequence, java.io.Serializable)

Interface **Serializable** declared methods for processing a string as a stream of characters (write string to file, ...)

The *String* Class

- The String class contains an **immutable** string (Once an instance is created, the string it contains cannot be changed) ← **No setter is implemented**
- **Almost of it's methods will return a new string.**
- Construct a string:

`String s1 = new String("immutable");`

`String s2= new String (new char[] {'a', 'b', 'c'});`

or

`String s3 = "immutable";`

String pool

```
public class StringDemo {
    public static void main (String[] args)
    {
        String s1="Hello"; // string pool
        String s2="Hello"; // string pool
        System.out.println("s1==s2: " + (s1==s2));
        String s3= new String("Hello");
        String s4= new String("Hello");
        System.out.println("s3==s4: " + (s3==s4));
        System.out.println("s3 equals s4: " + (s3.equals(s4)));
        String s5= new String ( new char[] { 'H', 'E', 'L', 'L', 'O' });
        System.out.println("s3 equals s5 ignoring case: " + (s3.equalsIgnoreCase(s5)));
        System.out.println(s5);
        s5= s5.toLowerCase();
        System.out.println(s5);
    }
}
```

String pool

Hello

s1

s2

Shallow comparing: Compare two references

Deep comparing: Compare two values

hello

HELLO

HELLO

garbage

s5

s5

String pool: a way to save memory

Output - Chapter08 (run)

```
run:
s1==s2: true
s3==s4: false
s3 equals s4: true
s3 equals s5 ignoring case: true
HELLO
hello
```

The *String* Class

Compare 2 strings: should use equals()

```
String st1 = "abc";
String st2 = "xyz";
if(st1.equals(st2)){
    ...
}
```

The *String* Class

Modifier and Type	Method and Description
char	<u>charAt</u> (int index)
char[]	<u>toCharArray</u> ()
byte[]	<u>getBytes</u> ()
int	<u>codePointAt</u> (int index), <u>compareTo</u> (<u>String</u> anotherString) <u>compareToIgnoreCase</u> (<u>String</u> str), <u>hashCode</u> (), <u>indexOf</u> (int ch), <u>indexOf</u> (...), <u>lastIndexOf</u> (...), <u>length</u> ()
<u>String</u>	<u>trim</u> (), <u>toString</u> (), <u>concat</u> (<u>String</u> str), <u>replace</u> (...), <u>replaceAll</u> (...) <u>replaceFirst</u> (...), <u>substring</u> (...), <u>toLowerCase</u> (...), <u>toUpperCase</u> (...)
static <u>String</u>	<u>copyValueOf</u> (...), <u>format</u> (...), <u>valueOf</u> (...)
boolean	<u>contains</u> (<u>CharSequence</u> s), <u>endsWith</u> (<u>String</u> suffix), <u>startsWith</u> (...), <u>isEmpty</u> (), <u>matches</u> (<u>String</u> regex) <u>equals</u> (<u>Object</u> anObject), <u>equalsIgnoreCase</u> (...)
void	<u>getChars</u> (int srcBegin, int srcEnd, char[] dst, int dstBegin)
<u>String</u> []	<u>split</u> (<u>String</u> regex), <u>split</u> (<u>String</u> regex, int limit)
<u>CharSequence</u>	<u>subSequence</u> (int beginIndex, int endIndex)

The *String* Class

```
import java.util.Scanner;

public class StringDemo {

    public static void main(String args[]){
        Scanner sc= new Scanner(System.in);
        String origin, replaced, replacement;
        System.out.print("Enter original string:");
        origin= sc.nextLine();
        System.out.print("Enter replaced string:");
        replaced= sc.nextLine();
        System.out.print("Enter replacing string:");
        replacement= sc.nextLine();
        origin = origin.replaceAll(replaced, replacement);
        System.out.println("After replacing:" + origin);
        System.out.println("Uppercase:" + origin.toUpperCase());
        System.out.println("Origin:" + origin);
        System.out.print("Enter the index of extracted character:");
        int index= Integer.parseInt(sc.nextLine());
        System.out.format("The %d(th) character:%c\n", index, origin.charAt(index));
    }
}
```

```
run:
Enter original string:do not love me
Enter replaced string:love
Enter replacing string:hate
After replacing:do not hate me
Uppercase:DO NOT HATE ME
Origin:do not hate me
Enter the index of extracted character:3
The 3(th) character:n
```

StringBuffer, StringBuilder Classes

- Java's **StringBuffer** and **StringBuilder** classes represent strings that can be dynamically modified.
 - StringBuffer is threadsafe.
 - StringBuilder (introduced in 5.0) is not threadsafe.
- Almost of their methods are the same as methods in the String class.

Thread: Unit of code (method) is running

Multi-threading program: A program has some threads running concurrently. If 2 threads access common data, their values are not unpredictable. So, in multi-thread programming, JVM supports a mechanism in which accesses to common resources must carry out in sequence based on synchronized methods.

Threadsafe class: A class with synchronized methods.

The *StringBuffer* - *threadsafe*

public final class **StringBuffer** extends Object
implements Serializable, CharSequence

```
public class StringBufferDemo {
    public static void main(String args[]){
        StringBuffer sBuf= new StringBuffer ("01234567");
        System.out.println(sBuf);
        sBuf.append("ABC");
        System.out.println(sBuf);
        sBuf.insert(2, "FAT PERSON");
        System.out.println(sBuf);
        sBuf.reverse();
        System.out.println(sBuf);
    }
}
```

run:

01234567

01234567ABC

01FAT PERSON234567ABC

CBA765432NOSREP TAF10

StringBuilder

public final class **StringBuilder** extends [Object](#)
implements [Serializable](#), [CharSequence](#)

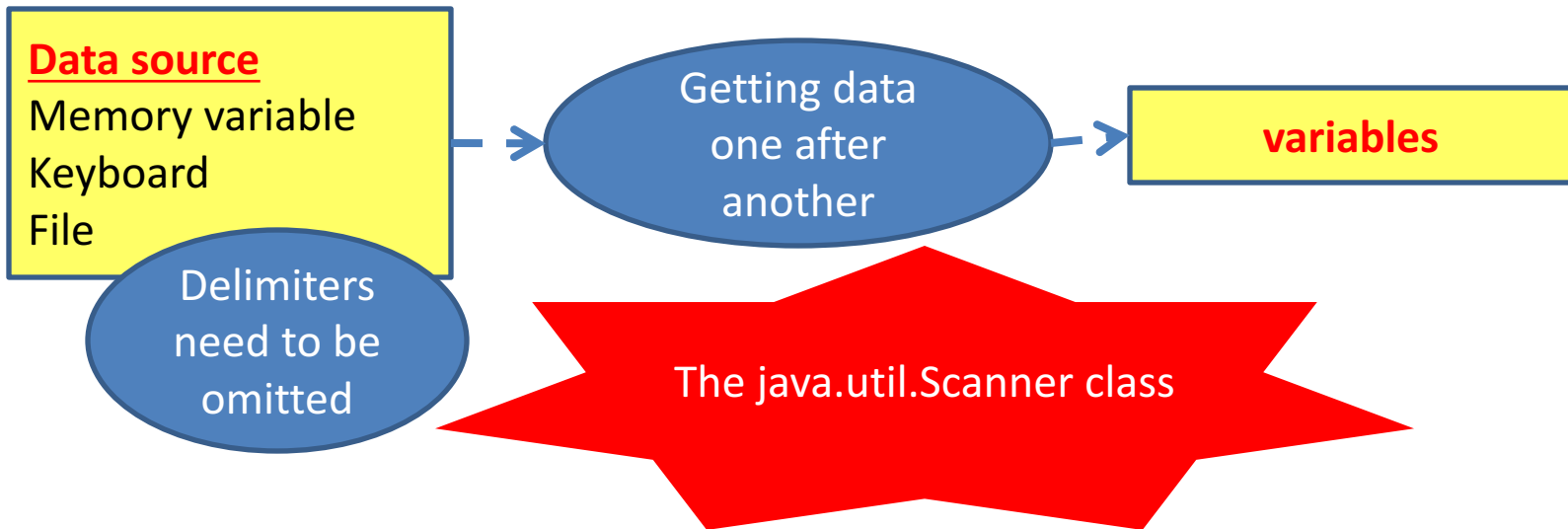
- The StringBuilder class was introduced in 5.0. It is nearly identical to StringBuffer.
- Major difference: string builders are **not threadsafe**.
- If you want multiple threads to have **concurrent access** to a mutable string, use a string buffer.
- If your mutable string will be accessed only by a single thread, there is an advantage to using a string builder, which will generally execute faster than a string buffer.

String Concatenation, the Easy Way

- 02 ways:
 - `String.concat()` method of the `String` class and the `StringBuffer.append()`.
 - Overloaded `+` operator.

Scanning Text

How to get data from a data source?

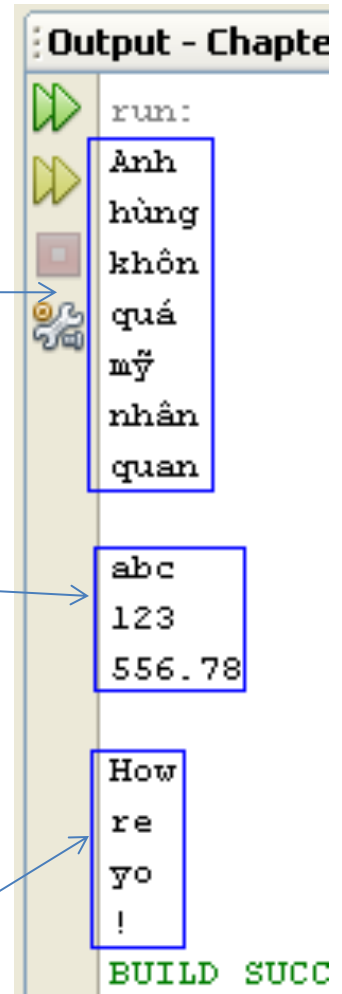


- **Class:** `java.util.Scanner`
- **Data in data source are characters**
- **Methods for getting data:** `next()`, `nextXXX()`
- **Methods for checking availability of data :** `hasXXX()`
- **Token:** group of characters that has a meaning.

Scanning data from a string

```
import java.util.Scanner;

public class ScannerDemo {
    public static void main(String[] args) {
        String S= "Anh hùng khôn quá mỹ nhân quan";
        Scanner sc1= new Scanner (S);
        while (sc1.hasNext()) System.out.println(sc1.next());
        System.out.println();
        String S2= "abc 123 556.78";
        Scanner sc2= new Scanner (S2);
        System.out.println(sc2.next());
        System.out.println(sc2.nextInt());
        System.out.println(sc2.nextDouble());
        System.out.println();
        String S3= "  How are      you!  ";
        String delim ="[au\\s]+"; // s:space, +: >=1 occurrence
        Scanner sc3= new Scanner (S3);
        sc3.useDelimiter(delim);
        while (sc3.hasNext()) System.out.println(sc3.next());
    }
}
```

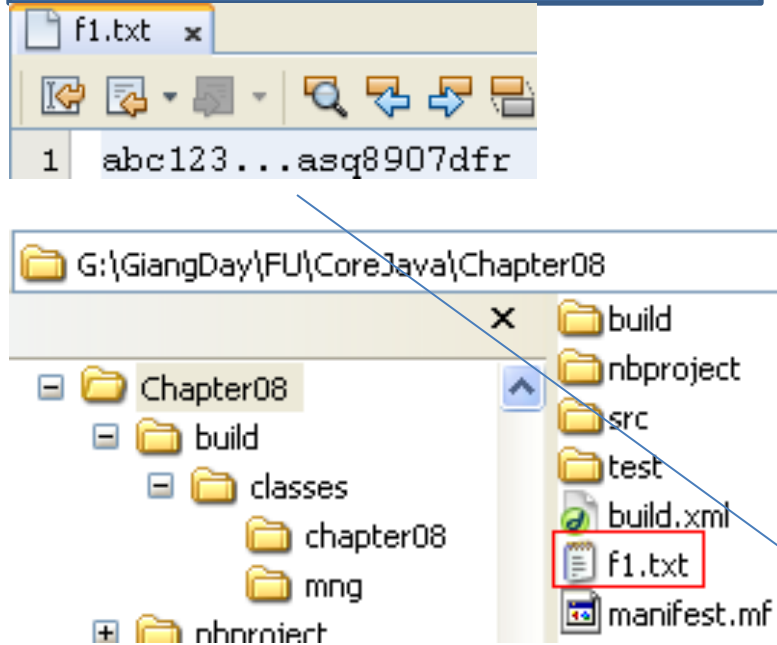


The default delimiter is the blank character. You can designate delimiters.
 [au\\s] means that a, u and space(\\s) are delimiters.
 + means that number of occurrences is equal or greater than 1

Scanning data from a text file

Specifying Delimiters

Add New File/ Other/ Empty File
File name: f1.txt



```
import java.io.File;
import java.util.Scanner;

public class ScannerDemo2 {
    public static void main (String[] args) {
        File f= new File ("f1.txt");
        try
        { Scanner sc= new Scanner(f);
          sc.useDelimiter("[[.]\d]+");
          while (sc.hasNext())
              System.out.println(sc.next());
        }
        catch (Exception e)
        { System.out.println(e);
        }
    }
}
```

Output - C

```
run:
abc
asq
dfr
```

Common Patterns:

[] : representing a character

. (arbitrary character)

\d (digits) \w (word characters) \s (space)

Quantifiers: * (>=0) , + (>=1) , ? (zero or one)

[a-zA-Z] : a character from a to z or A to Z

Read java documentation for more details

Class: java.util.regex.Pattern

Splitting a string into substrings

The method `split(delimiters)` of the class `String` and the `java.util.StringTokenizer` are used.

```
import java.util.StringTokenizer;
public class SplittingStringDemo {
    public static void main(String[] args) {
        String str = "I study hard. So, I pass the examination";
        // Using the method String[] split() of the String class
        String[] strs = str.split("[ ,.]+");
        for (String s:strs) System.out.println(s);
        System.out.println();
        // Using the java.util.StringTokenizer class
        StringTokenizer stk= new StringTokenizer(str,"[ ,.]");
        System.out.println("Number of substrings: " + stk.countTokens());
        while (stk.hasMoreTokens())
            System.out.println(stk.nextToken());
    }
}
```

Output - Chapter08 (run)

```
run:
I
study
hard
So
I
pass
the
examination

Number of substrings: 8
I
study
hard
So
I
pass
the
examination
BUILD SUCCESSFUL (total
```

Formatting Output

%[argument_index\$][flags][width][.precision]conversion

See API documentation for more details (api/java/util/Formatter.html#syntax).

```
public class PrintWithFormat {
    public static void main (String[] args){
        String S="Hello";
        int i=5;
        long l=58;
        float f= 7.2f;
        double d=8.9;
        boolean b= true;
        char c='A';
        System.out.format("%s,%2d,%4Xh,%5.2f,%10.3f,%3c,%b\n",S,i,l,f,d,c,b );
        System.out.format("%3$3b,%1$3d,%2$12s\n", i,S,b);
        }
}
```

1 2 3

run:

```
Hello, 5,  3Ah, 7.20,      8.900,  A,true
true,  5,      Hello
```

Formatting Output

```
import java.text.DecimalFormat;

public class DecimalFormatDemo {

    static public String customFormat(String pattern, double value ){
        DecimalFormat myFormatter = new DecimalFormat(pattern);
        String output = myFormatter.format(value);
        return output;
    }

    static public void main(String[] args) {
        System.out.println(customFormat("###,###.###", 123456.789));
        System.out.println(customFormat("###.##", 123456.789));
        System.out.println(customFormat("000000.000", 123.78));
        System.out.println(customFormat("$###,###.###", 12345.67));
    }
}
```

run:

123,456.789

123456.79

000123.780

\$12,345.67

Summary

- **Working with Numbers:**
 - Wrapper classes: Number, Character
 - The java.lang.Math class
 - Autoboxing and unboxing .
- **The java.lang.Math**
- **String class:**
 - Create and manipulate strings.
 - Compares the String and StringBuilder classes.
- **Scanning Text**
- **Formatting output**