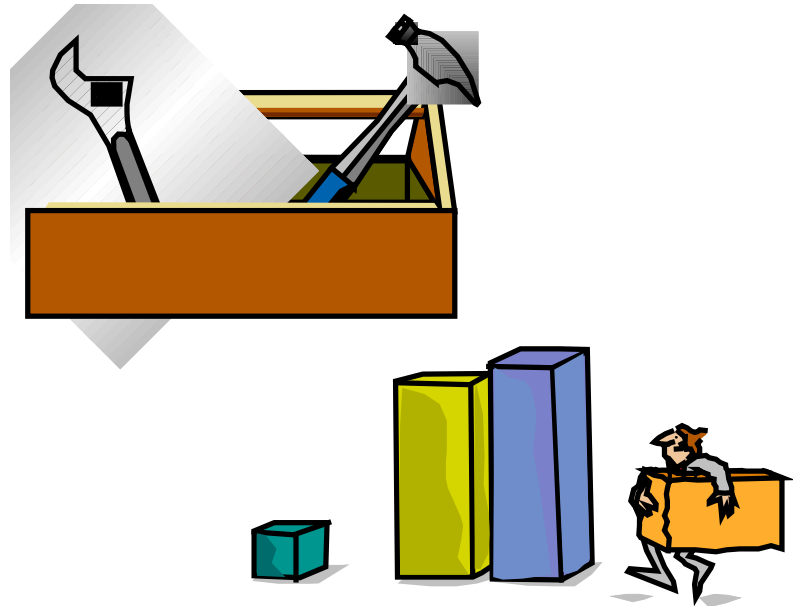


Java 工具类



- **Java 工具包**

- **Java.lang package**

- **Object、Wrapper Classes for Primitive Types、Math、String、System、Runtime、Thread、Class、Exception、Process**

- **java.util package**

- **GregorianCalendar、data-structure classes、Random、StringTokenizer、Observable**

- **Java 数组**

- **Java Applet**

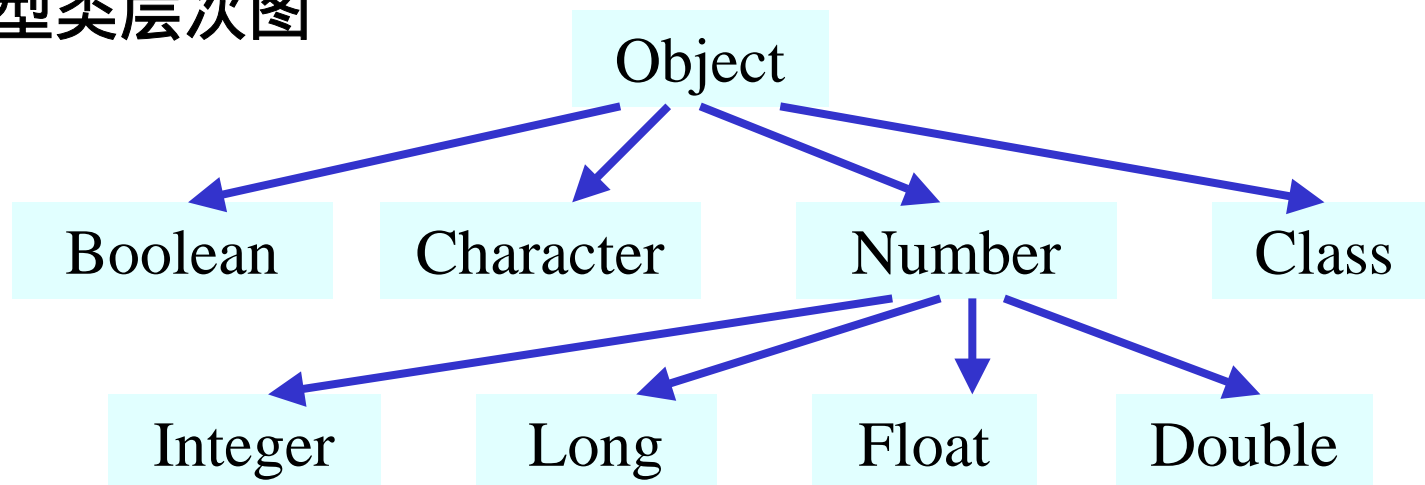


方法:

- **Object clone():**[return a new copy of this object]
- **boolean equals(Object):** [compares object]
- **void finalize():**[do nothing]
- **final Class getClass():** [object contains run-time type information]
- **int hashCode():**[returns an integer suitable for determining the hash code for an object]
- **String toString()**



- 数据类型类层次图



- 构造函数

- 依据基本类型并创建该类型对象的构造函数
- 对单一string参数解码以得到该对象初始值的构造函数

- 基本方法

- toString
- typeValue:Character.charValue
- equals
- hashCode



- **Constants**

- **MAX-VALUE、 MIN -VALUE**

- **Constructor**

- **public Integer (int value)**

- **public Integer(String s)**

- **Methods**

- **public double doubleValue ()**

- **public int intValue()**

- **public long longValue ()**



•Methods

➤ **public static String toString()**

➤ **public static int parseInt(String s)**

int j = Integer.parseInt("123");

➤ **public static Integer valueOf(String s)**

int j = Integer.valueOf("123").intValue();

➤注：类似double、float数据，没有parseInt方法，只能用上述步骤将字符串转换成数值数据

float f = Float.valueOf("12.3").floatValue();



Math 类包括一系列常量和常用的数学运算方法

- 所有的操作都是 *double*
- **Math.E** 代表 $e(2.7182818284590452354)$
Math.PI 代表 (3.14159265358979323846)
- 在所有方法中角度都是采用弧度制，所有参数和返回值都是 *double*
- 静态方法 **random()** 返回伪随机数 r ($0.0 \leq r < 1.0$)
- 例如：`int j = (int)(Math.random()*10)+1;`



标准的 I/O Stream

- **public static InputStream in**
- **public static PrintStream out**
- **public static PrintStream err**

System 方法

- **Public static long currentTimeMillis()**
- **Public static void exit(int status)**
- **Public static void gc()**



- **Java 语言中数组的特点**
 - 数组中元素是有先后次序的
 - 每个数组中元素类型是相同的
 - 通过索引直接访问元素
 - 数组一旦建立大小就固定了



```
String[ ] friends;     //声明数组
```

```
friends = new String[3];   //创建数组空间
```

```
// 初始化数组
```

```
friends[0] = “Greg”;
```

```
friends[1] = “Sam”;
```

```
friends[2] = “Elmo”;
```



- 声明数组的名称和数组所包含的数据类型或元素的类名

<type>[] *identifier*

or <type> *identifier* []

- 示例: **int[] ages; int ages[]**

Date dateArray[];



- 在声名时不能确定
- 初始化时定义并固定
- 通过 `<var>.length` 获得
- 索引: `0 - length-1`,

`<array_var>[< expression>]`

- 异常 : `IndexOutOfBoundsException`
- 示例 : `int name[50];` **X**



- 数组通过new创建数组空间
- 创建数组必须指明数组的长度:
arrayName = new type[arraysize];
- 示例:
int[] ages; ages = new int[3];
Date dateArray = new Date[10];



- 如果数组元素的类型是基本数据类型，每个元素会被自动初始化
- 如果数组元素的类型是某个类的对象，则有两种初始化方式：
 - 创建后初始化（两步空间分配）
 - `Date dateArray[] = new Date[10];` //new reference
 - `Date[0] = new Date(paralist); ...// new element`
`Date[9] = new Date (paralist);`
 - 定义数组的同时对数组初始化

type arrayName[] = {element1[, element2...]};

`Int myArray[] = {1,2,3,4} //no size of array`



- 引用方式

arrayName[index]

Index : 0 to capacity – 1

- 例如:

```
int[ ] primes = {2, 3, 5, 7, 11, 13, 19};
```

```
int x = primes[3];
```

```
int y = primes[4];
```



- 复制数组变量

- 例如

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

```
int another[ ] = myArray;
```

- 复制数组元素

- System.arraycopy(from, fromIndex, to, toIndex, count)

- 例如

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

```
int another[ ] = new int[4];
```

```
Sysytem.arraycopy(myArray,0,another,0,4);
```



从一个方法里返回一个数组

```
Public class ArrayReturn {
    public int[ ] returnsArray(boolean flag) {
        int[ ] array1 = {1,2,3,4,5,6};
        int[ ] array2 = {10,20,30};
        if (flag) { return array1; }
        else { return array2; } }
    public static void main(String[ ] args){
        ArrayReturn x = new ArrayReturn();
        System.out.println(x.returnsArray(true).length);}}
```



- 类Vector实现动态地分配对象列表
- 合适使用向量的情况：
 - 需要处理的对象数目不定，序列中的元素一定是对象
 - 需要将不同类的对象组合成一个数据序列
 - 需要做频繁的对象序列中元素的插入和删除
 - 经常需要定位序列中的对象或其他查找操作
 - 在不同的类之间传递大量的数据
- 创建向量对象

```
public Vector(int initCapacity, int capacityIncrement)  
    Vector v = new Vector(100,50);
```



- **插入元素**
 - **void addElement(Object obj)**
 - **void insetElement(Object obj, int index)**
- **修改或删除元素**
 - **void setElementAt(Object obj, int index)**
 - **boolean removeElement (Object obj)**
 - **void removeAllElements()**
- **查找元素**
 - **Object elementAt(int index)**
 - **boolean contains(Object obj)**
 - **int indexOf(Object obj, int start-index)**
 - **int lastIndexOf(Object obj, int start-index)**



- 在Java中字符串作为对象处理，类String和类StringBuffer都可以表示字符串
- 字符串常量: anonymous string objects
`String color="blue";`
`int len = "blue".length();`
- Classes: extends `java.lang.Object`
`String`: for string constants
`StringBuffer`: for string variables(resizable)



- **public *final* class java.lang.String extends java.lang.Object**
- **构造函数**
 - (1) **public String():** construct a null string “”
 - (2) **public String(char chars[], int offset, int count)**
 - (3) **public String(char chars[])**
 - (4) **public String(byte bytes[], int hibyte, int offset, int count)**
 - (5) **public String(byte bytes[], int hibyte) (*not useful*)**
 - (6) **public String(String value)**
 - (7) **public String(StringBuffer buffer)**



```
char chararray[]={ 'b','i','r','t','h', ' ', 'd', 'a', 'y' };
byte bytearray[]= { 'n', 'e', 'w', ' ', 'y', 'e', 'a', 'r' };
StringBuffer buffer;
String s, s1, s2, s3, s4, s5, s6, s7;
public void init(){
    s = new String("hello");
    buffer = new StringBuffer();
    buffer.append("Welcome");
    s1 = new String("");                // a null string
    s2 = new String(s);                 // hello
    s3 = new String(chararray);         //birth day
    s4 = new String(chararray, 6, 3);   //day
    s5 = new String(bytearray, 0, 4, 4); // year
    s6 = new String(bytearray, 0);      // new year
    s7 = new String(buffer);            // welcome    }
```



(1) `public int length()`

- 返回当前字符串中字符的个数
- Example: `s.length();` // not `s.length`

(2) `public char charAt(int index)`

选择并返回字符串中该位置上的字符

- 例如：//计算一个字符串中每种字符的数目

```
for (int i = 0; i < str.length(); i++)
```

```
    counts[str.charAt(i)]++;
```



•判断字符串的前缀和后缀

- `public boolean startsWith (String prefix)`
- `public boolean endsWith (String suffix)`

•字符串中单个字符的查找

- `public int indexOf(int ch)`

Example: letters.indexOf((int)'c')

- `public int indexOf(int ch, int fromIndex)`
- `public int lastIndexOf(int ch)`
- `public int lastIndexOf(int ch, int fromIndex)`



•字符串中子串的查找

- `public int indexOf(String str)`
- `public int indexOf(String str, int fromIndex)`
- `public int lastIndexOf(String str)`
- `public int lastIndexOf(String str, int fromIndex)`

•返回字符串中的子串

- `public String substring(int beginIndex, int endIndex)`

根据字符串中的位置返回一个新字符串 (`beginIndex` 到 `endIndex - 1`)



```
Public static String quotedString(String frm,char start,
                                char end) {
    int startPos = from.indexOf(start);
    int endPos = from.lastIndexOf(end);
    if (startPos == -1)
        return null;
    else if (endPos == -1)
        return from.substring(startPos);
    else
        return from.substring(startPos, endPos + 1);
}
```



• 比较两个字符串

➤ **public int compareTo (String aString)**

s1.compareTo(s2) returns

0

if s1=s2;

a negative number

if s1 is less than s2;

a positive number

if s1 is greater than s2

➤ **public boolean equals(Object anObject)**

与“==”不同

➤ **public boolean equalsIgnoreCase(String aString)**

two characters(c1,c2) are equal only when:

(a) c1==c2; OR

(b) c1.toUpperCase()= c2.toUpperCase(); OR

(c) c1.toLowerCase()= c2.toLowerCase();



```
private String[] table;  
public int position(String key) {  
    int lo = 0;  
    int hi = table.length - 1;  
    while (lo <= hi) {  
        int mid = lo + (hi - lo)/2;  
        int cmp = key.compareTo(table[mid]);  
        if (cmp == 0)  
            return mid;  
        else if (cmp < 0)  
            hi = mid - 1;  
        else  
            lo = mid + 1;  
    }  
    return -1;  
}
```



```
s1= new String(“hello”);  
s2 = “hello”;  
s3 = “Hello”;
```

• s1.equals (“hello”)	true
• s1.equals (“Hello”)	false
• s1 == “hello”	false
• s2 == “hello”	true
• s1==s2	false
• s2. equalsIgnoreCase (s3)	true



•连接字符串

➤ **public String concat(String str)**

• “concat” 类似 ‘+’

newS=oldS concat(“(not”) == newS=oldS+“(not”

•创建字符串对应的字符数组

➤ **public char[] toCharArray()**



```
Public static String squeezeOut(String from, char toss)
{
    char []chars = from.toCharArray();
    int len = chars.length;
    for (int i = 0; i < len; i ++)
    {
        if (chars[i] == toss) {
            - -len;
            System.arraycopy(chars, i +1, chars, i, len-i);
            - - i; }
    }
    return new String( chars, 0, len);
}
```



- 特点

- String 类的对象: 只读
- StringBuffer类的对象: 可修改,用于实现 + and +=
- java.lang.StringBuffer extends java.lang.Object

- 构造函数

- (1) public StringBuffer()

- 建立一个空串(初始容量为 16)

- (2) public StringBuffer(int length)

- 建立一个长度给定的空串

- (3) public StringBuffer(String str)

- 建立一个包含字符串对象str的字符串变量, 初始容量为 str.length()+16



- **public int length()**
- **public int capacity()**

```
Class StringBufferDemo {  
    public static void main(String args[]) {  
        StringBuffer sb = new StringBuffer("Hello");  
        System.out.println("length =" + sb.length());  
        System.out.println("capacity =" + sb.capacity());  
    } }
```



- **public StringBuffer append(参数对象类型 参数对象名)**
- **public StringBuffer insert(int 插入位置, 参数对象类型 参数对象名)**
- **public char charAt (int index)**
- **public void setCharAt (int index, char ch)**
- **“+”或“+=”运算**

➤ **”BC”+22**

new StringBuffer().append (“BC”).append(22).toString()

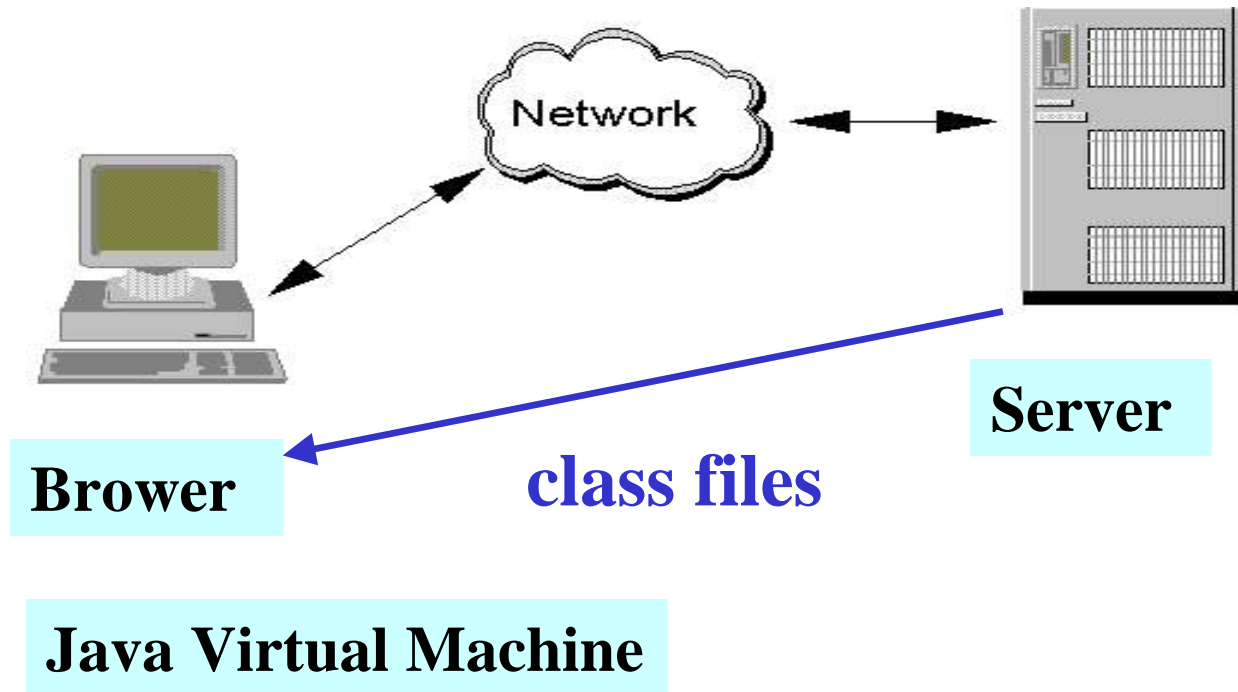
➤ **s+=”!”**

new StringBuffer().append(s). append(“!”).toString()

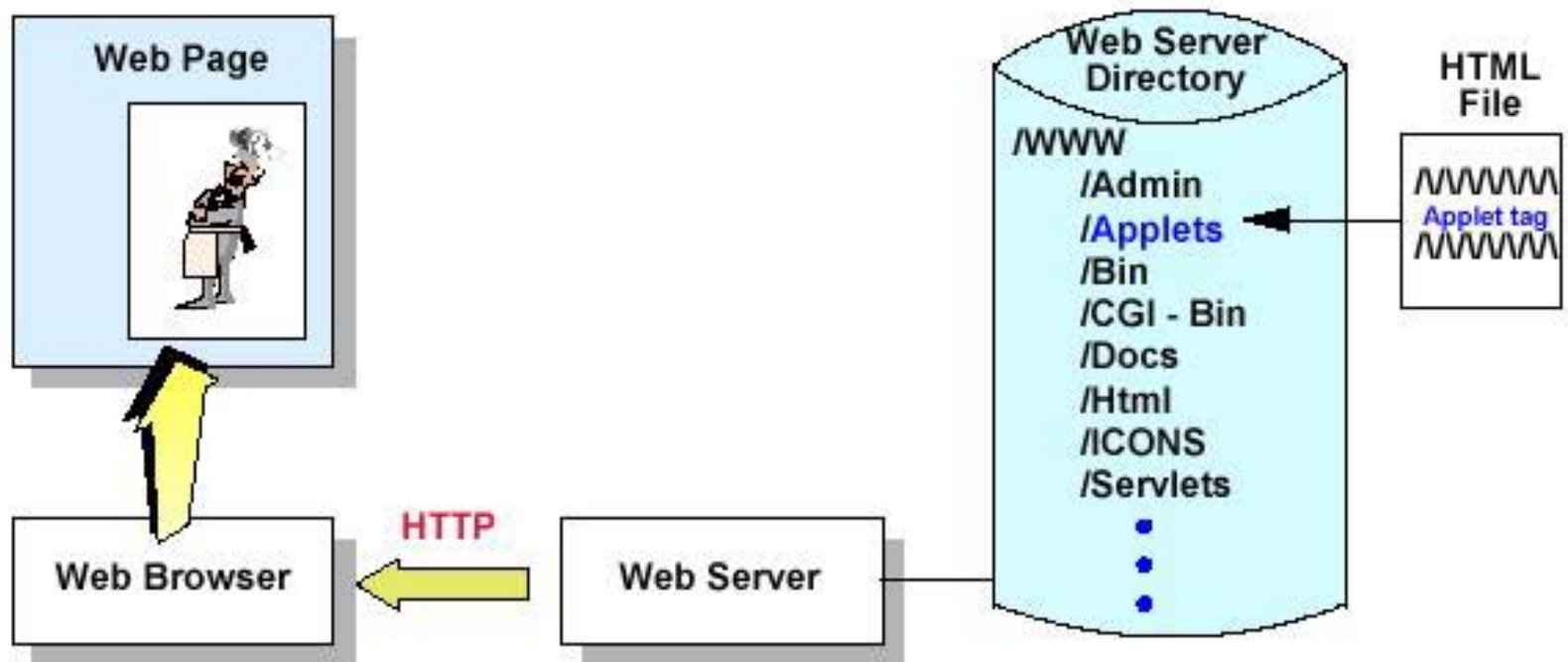


```
Public static void replace(StringBuffer str, char from,
char to) {
    for (int i = 0; i < str.length(), i++)
        if (str.charAt(i) == from)
            str.setCharAt(i, to);
}
```

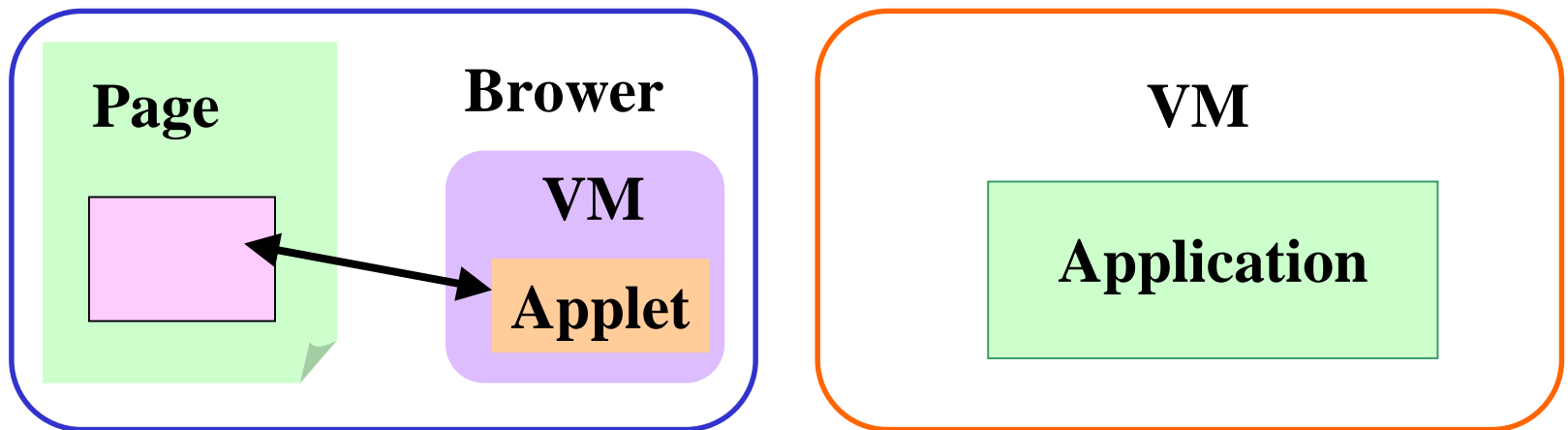




How Java Applets Work



- **Applet**
 - 嵌入在 Web 页面内并且运行在 支持Java的 Web browser
- **Application**
 - 单独运行在JVM中的Java 程序



特性	Application	Applet
运行程序	Java classFileName (from a command prompt)	Browser Web Page Browser Open HTML File Appletviewer htmlFileName
参数	命令行参数 (args[0]...)	在HTML文件中 (PARAM...)
是否有 main()	YES	NO
支持GUI	YES	YES
安全性	NO	YES



- Applet creation

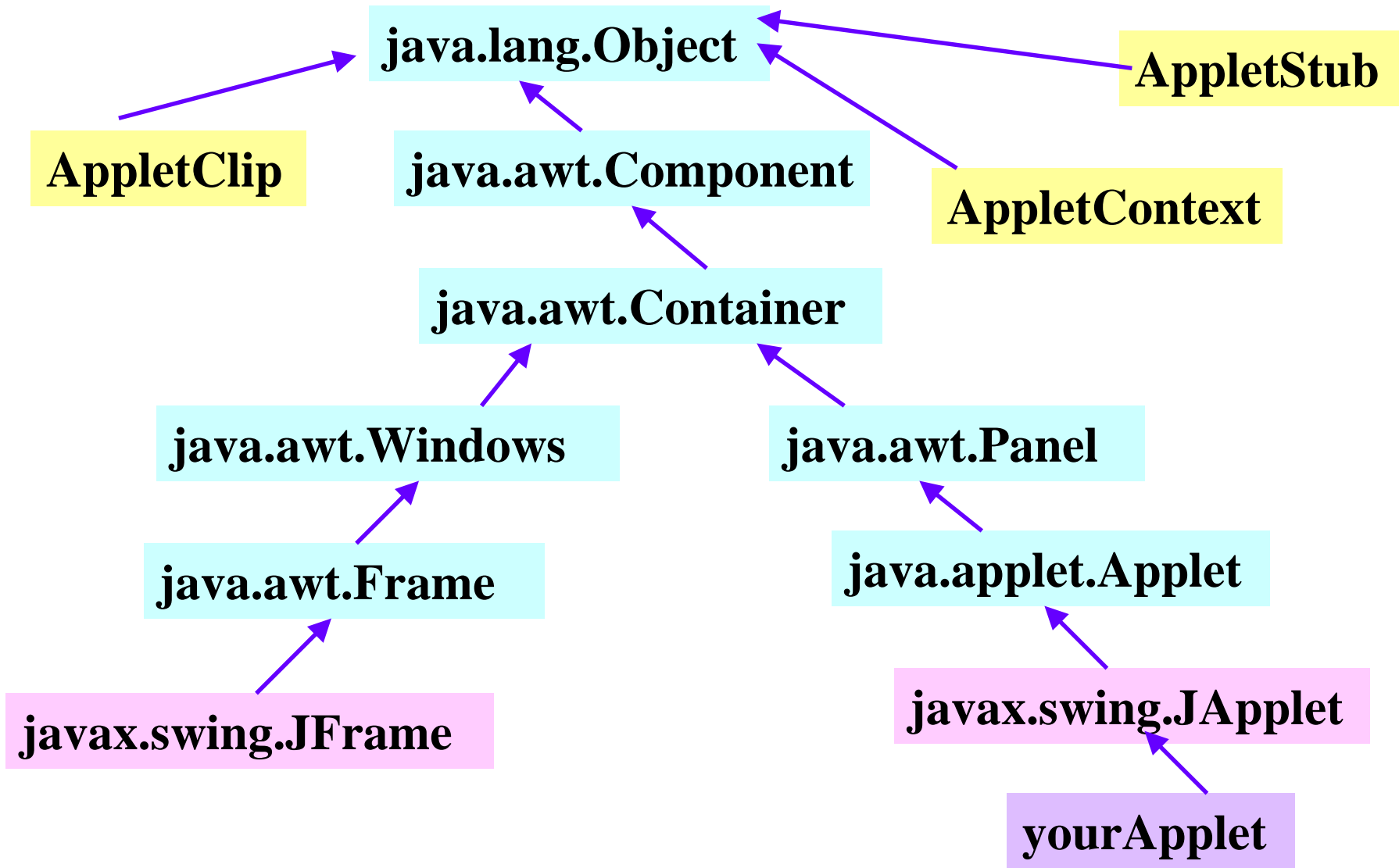
```
import java.applet.*;  
public class Hello extends Applet{ ...}
```

- JApplet creation

```
import javax.swing.*;  
public class Hello extends JApplet{ ...}
```

- 支持 Java 的浏览器 (HotJava、Netscape Navigator、IE)
- Applet viewer (\jdk\bin\)
 - Applet viewer [option] urls...
 - 只能读类似 OBJECT、EMBED、APPLET (can't read other HTML tags) 的标记





Each Applet inherits from `java.applet.Applet`/`javax.swing.JApplet`



- **init()**
 - 完成主类实例的初始化工作
- **start()**
 - 启动浏览器运行Applet的主线程
 - 可多次启动：init()后自动调用、使用浏览器的Reload操作、浏览器从其他HTML页面返回
- **stop()**
 - 暂停执行Applet的主线程
- **destroy()**
 - 消灭Applet实例并关闭浏览器

• [Example : life.java](#)

• **Example : LifeCycle.java P130-131**



- **paint()方法**：在Applet的界面中显示文字、图形和其他界面元素
- **调用paint()方法的事件**：
 - Applet启动后，自动调用paint()重新描绘自己的界面
 - Applet所在的浏览器窗口改变后又重新显示在屏幕的最前方时自动调用paint()方法
 - Applet的其他相关方法被调用时，系统也会响应地调用paint()方法
- **paint()方法参数**：Graphics类的对象g

当一个Applet类实例被初始化并启动时，浏览器将自动产生一个Graphics类的实例g，并作为参数传递给Applet类实例的paint()方法



- 载入一个 Applet

调用 *constructor* 产生一个实例

Browser invokes *init()* method

Browser invokes *start()* method

- 离开并返回某页面

Browser invokes *stop()* method

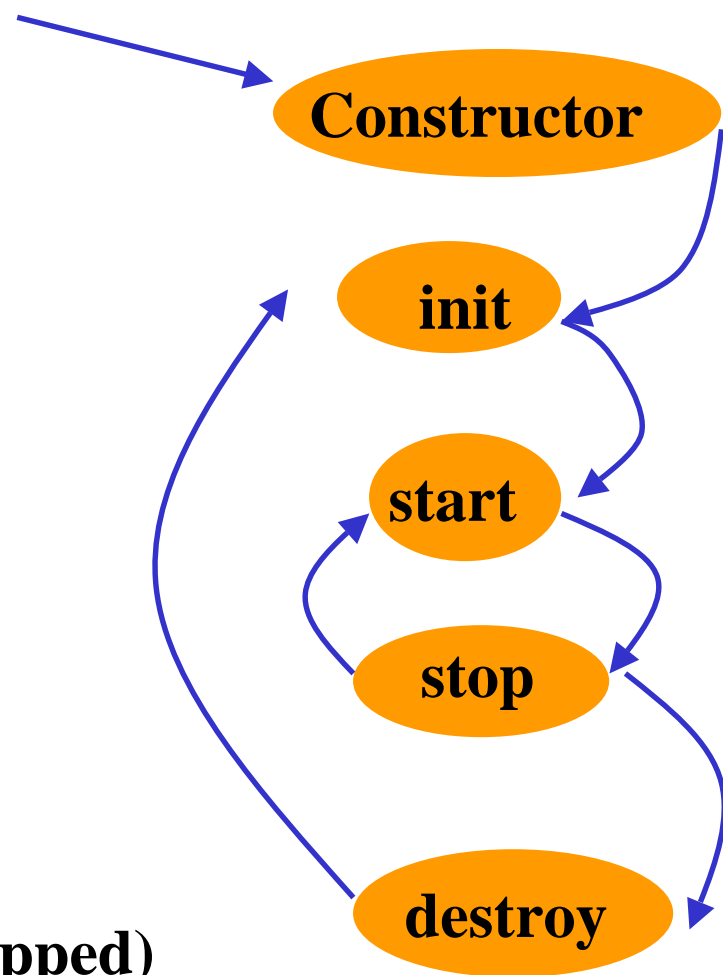
Browser invokes *start()* method

- 重新载入一个页面

stop() - suspend any activity

destroy() - free resources (threads stopped)

Constructor-Create new Applet (*init()* *start()*)



- Applet's Tag in HTML (必备的)

```
<APPLET CODE=  
        CODEBASE=  
        WIDTH=  
        HEIGHT= >  
  
</APPLET>
```

- Applet的三个参数
 - 类文件的名字
 - 类文件的地址
 - Applet 在页面中的表示



其他的Applet标记

<APPLET

[CODEBASE = codebaseURL]

CODE=appletClassFile

[ALT=alternativeText]

[NAME=appletInstanceName]

WIDTH = pixels HEIGHT = pixels

[ALIGN = alignment]

[VSPACE = pixels] [HSPACE = pixels]>

[<PARAM NAME=AttrName VALUE =AttrValue>]

[<PARAM NAME=AttrName2 VALUE =AttrValue>]

..... [HTML displayed in the absence of Java]

</APPLET>



- **ALIGN=alignment (LEFT、BOTTOM、TEXTTOP...)**
- **CODEBASE**
 - 类文件的目录 (URL)
 - 如果缺省, 使用 HTML 文件目录
 - class and HTML 文件可以在不同的主机上
- **ALT=alternateAppletText**
 - 当浏览器能认识该标记但不能运行applet 时显示的文字
- **NAME**
 - 有两个或两个以上的 applet 时用于识别
 - 使用类 AppletContext 中的getApplet() 方法获得 name



- **ARCHIVE**

- 列出 Java jar 文件,包括类和其它资源

- 在 Java1.1中使用 Swing;

- 从网络上下载Swing 补丁(swing.jar)

<APPLET

CODE = “CalculatorApplet.class”

**ARCHIVE = “CalculatorClasses.jar,
swing.jar”**

WIDTH = 100

HEIGHT = 150

\APPLET >



- Applet 可以从 html 文件中获得参数
- Tag PARAM
 - HTML文件
`<PARAM NAME="Param_Name" VALUE="">`
 - APPLET文件
`String getParameter("Param_Name")`
- [Example](#)(DemoOfParameter.html)

HTML

```
<APPLET CODE="Demo.class" WIDTH=400 HEIGHT=300>  
<PARAM NAME="font" VALUE="TimesRoman">  
<PARAM NAME="size" VALUE="20">  
</APPLET>
```



Java Code

```
public class Demo extends Applets {  
    public void paint(Graphics g){  
        String fontName=getParameter("font");  
        String sizeName=getParameter("size");  
        int fontSize=Integer.parseInt(sizeName);  
        Font fnt=new Font(fontName,Font.BOLD,fontSize);  
        g.setFont(fnt);  
        g.drawString("Demo",40,50);  
    }  
}
```



Parameter Reconsidered: default

```
String sizeName=getParameter("size");  
int fontSize=20; //default  
if (sizeName!=null)  
    int fontSize=Integer.parseInt(sizeName);
```



- **P181 : 6-8、 6-11、 6-13、 6-14、 6-16、 6-17**

