

Advanced Java Programming

Ohad Barzilay,
Tel-Aviv University
Spring '06

Java Syntax & Structures

Identifiers, Keywords, Literals
Separators and Operators

Java Syntax Basics

- The java syntax is case-sensitive.
- Semi-colon (;) is used as line terminator.
- Curly braces ({, }) are used for block structure .
- There are several reserved keywords.

Language Syntax Symbols Examples

□ Literals (numbers, strings etc.):

■ 219 "hello"

□ Operators (arithmetic, boolean, etc.):

■ + - * / & |

□ Identifiers (names for variables, methods, classes etc.):

■ result

□ Keywords & Separators:

■ for if while { } () ;

Keywords

- The following keyword are reserved and cannot be used as identifiers:

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

Literals

- A literal is a source code representation for a value of:
 - a primitive type (Integer, Floating-Point, Boolean, Character)
 - the String type,
- Literals are the representation of simple data values in our language: mainly numbers and text.
- Reserved literal words: `null`, `true`, and `false`

Integer Literals

- An integer literal may be expressed in:
 - decimal (base 10),
 - hexadecimal (base 16), using the prefix 0x
 - octal (base 8), using the prefix 0

Floating-Point Literal Parts

- a whole-number part: -22
- a decimal point (represented by an ASCII period character): -22.
- a fractional part: -22.58
- an exponent (indicated by the letter e or E followed by an optionally signed integer): -22.58e3
- and a type suffix: -22.58e3f

$$= -22.58 * 10^3$$

Character Literals

- A character literal is expressed as a character or an escape sequence, enclosed in single quotes (').
- The single-quote, or apostrophe, character is `\u0027`
- A character literal is always of type `char`:
`'a'` `'%'` `'\t'` `'\\'`
`'\u03a9'` `'\uFFFF'` `'\177'`

String Literals

- A string literal consists of zero or more characters enclosed in double quotes. A string literal is always of type String:

`""` // *the empty string*

`"This is a string"` // *a string containing 16 characters*

`"This is a " +` // *a string-valued constant expression,*

`"two-line string"` // *formed from two string literals*

String Conversions

Type	To String	From String
<i>boolean</i>	<i>String.valueOf(boolean)</i>	<i>new Boolean(String).booleanValue()</i> OR <i>Boolean.parseBoolean(String)</i>
<i>int</i>	<i>String.valueOf(int)</i>	<i>Integer.parseInt(String, int base)</i> OR <i>new Integer(String).intValue();</i>
<i>long</i>	<i>String.valueOf(long)</i>	<i>Long.parseLong(String, int base)</i>
<i>float</i>	<i>String.valueOf(float)</i>	<i>new Float(String).floatValue()</i>
<i>double</i>	<i>String.valueOf(double)</i>	<i>new Double(String).doubleValue()</i>

String Conversions

- Converting primitive to Strings can also be done using the + operator:

```
int i = 4;
```

```
String s = "" + i;
```

- Converting strings to primitives can be done either by using the `static` wrapper methods, or by creating a temp wrapper object and apply the appropriate instance method

Boolean Literals

- The `boolean` type has two values, represented by the literals `true` and `false`.
- A Boolean literal is always of type `boolean`.

Separators

- The following nine ASCII characters are the separators (punctuators):
 - () { } [] ; , .
- Separators have special meanings in our language and so cannot be used freely (e.g. as part of an identifier).

Operators

Operators	Associative
<code>++ -- + unary - unary ~ ! (<data_type>)</code>	R to L
<code>* / %</code>	L to R
<code>+ -</code>	L to R
<code><< >> >>></code>	L to R
<code>< > <= >= instanceof</code>	L to R
<code>== !=</code>	L to R
<code>&</code>	L to R
<code>^</code>	L to R
<code> </code>	L to R
<code>&&</code>	L to R
<code> </code>	L to R
<code><boolean_expr> ? <expr1> : <expr2></code>	R to L
<code>= *= /= %= += -= <<= >>= >>>= &= ^= =</code>	R to L

Arithmetic operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder
++	Increment
--	Decrement

Boolean Operators

<i>Operator</i>	<i>Use</i>	<i>Returns true if</i>
<code>&</code>	<code>exp1 & exp2</code>	<code>exp1 and exp2 are both true</code>
<code>&&</code>	<code>exp1 && exp2</code>	<code>exp1 and exp2 are both true, conditionally evaluates exp2</code>
<code> </code>	<code>exp1 exp2</code>	<code>either exp1 or exp2 is true</code>
<code> </code>	<code>exp1 exp2</code>	<code>either exp1 or exp2 is true, conditionally evaluates exp2</code>
<code>^</code>	<code>exp1 ^ exp2</code>	<code>if exp1 or exp2 is true but not both</code>
<code>!</code>	<code>!exp</code>	<code>exp is false</code>

Bit-wise Operators

- Perform the operation on each bit

Operator	Use	Evaluates
<code>~</code>	<code>~exp1</code>	<i>Flips all the bits of exp1</i>
<code>&</code>	<code>exp1 & exp2</code>	<i>Always evaluates exp1 and exp2</i>
<code> </code>	<code>exp1 exp2</code>	<i>Always evaluates exp1 and exp2</i>

- `~10000101 == 01111010`
- `10000101 | 10110001 == 10110101`
- `10000101 & 10110001 == 10000001`

Strong Typing

- ❑ The Java programming language is a strongly typed language.
- ❑ Strongly typed - every variable and every expression has a type that is known at compile time (as opposed to runtime).

Typing Features

- Limit the values
 - That a variable can hold or
 - That an expression can produce
- Limit the operations supported on those values
- Determine the meaning of the operations

Strong typing helps detect errors at
compile time

Java Types

- The data types of the Java programming language are divided into two categories:
 - primitive types
 - reference types



Reference Type

- Enables a mechanism to create and add new types using classes and objects.

- (Much) more later with classes...

Primitive Types

- A primitive type is predefined by the Java programming language and named by its reserved keyword.
- The primitive types are:
 - numeric
 - boolean
 - char

Java Numeric Types

- The integer types represent whole number:

byte, int, short, long

- The floating-point types represent real numbers:

float, double

Integer Type Ranges

- The four separate integer primitive data types ranges and number of bits are:

<u>Type</u>	<u>Storage</u>	<u>Min Value</u>	<u>Max Value</u>
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	-2^{63} or $< -9 \times 10^{18}$	$2^{63}-1$ or $> 9 \times 10^{18}$

Floating-point Type Ranges

- The two floating-point primitive data types ranges and number of bits are:

<u>Type</u>	<u>Storage</u>	<u>Approximate Min Value</u>	<u>Approximate Max Value</u>
float	32 bits	-3.4×10^{38}	3.4×10^{38}
double	64 bits	-1.7×10^{308}	1.7×10^{308}

- Only a very small sub-set of real numbers!

Unicode Character Type

- ❑ A char value stores a single character from the Unicode character set.
- ❑ The values range is between 0 and 65535.
- ❑ The amount of memory it requires: 16 bit or 2 bytes.
- ❑ The Unicode character set uses 16 bits per character.

Constants

- A constant is an identifier that is similar to a variable except that its value cannot be changed . You cannot assign to a constant!
- The final modifier is used for constant declaration:
 - `final <type> <Variable_name> = <value>;`
- Example:
 - `final int MAX_GRADE = 100;`

Statements

- Statements are the basic building block of any program – they are the ‘instructions’ that tell the computer what to do.
- “Print the results of the math”: *Signifies ‘end of statement’*



Statements in Methods*

- A method contains one or more statements.
 - `System.out.println("How are you?");`

* Method == Function

The main Method

- The main method is the ‘entry point’ to the program. It contains the code to be executed when the program starts.
- There can be only one main method per file.
- The main method has a standard syntax:

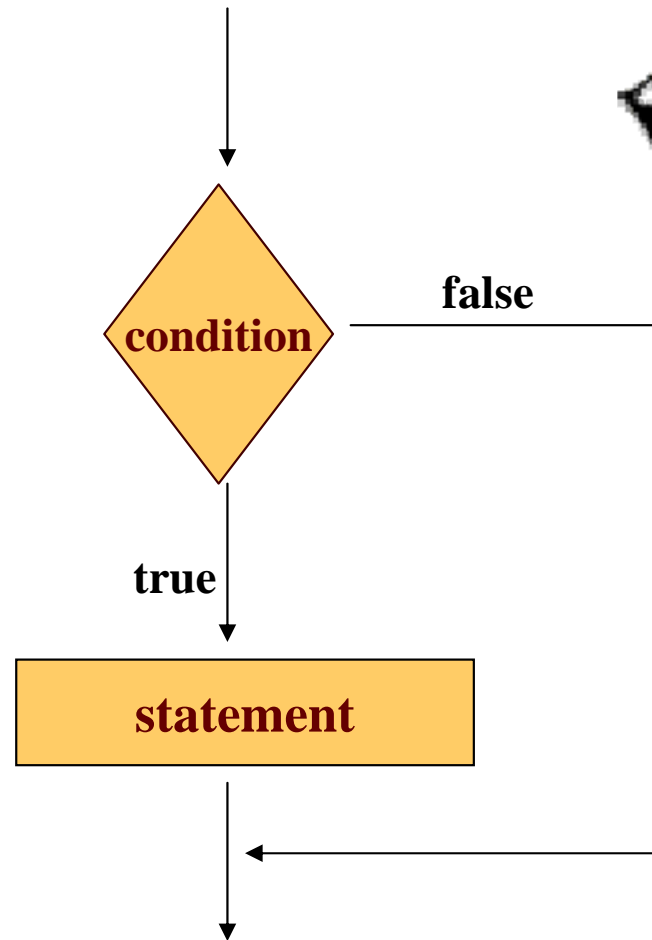
```
public static void main(String[] args){  
    // your code...  
}
```

Control Structures and statements

If Statement

Syntax:

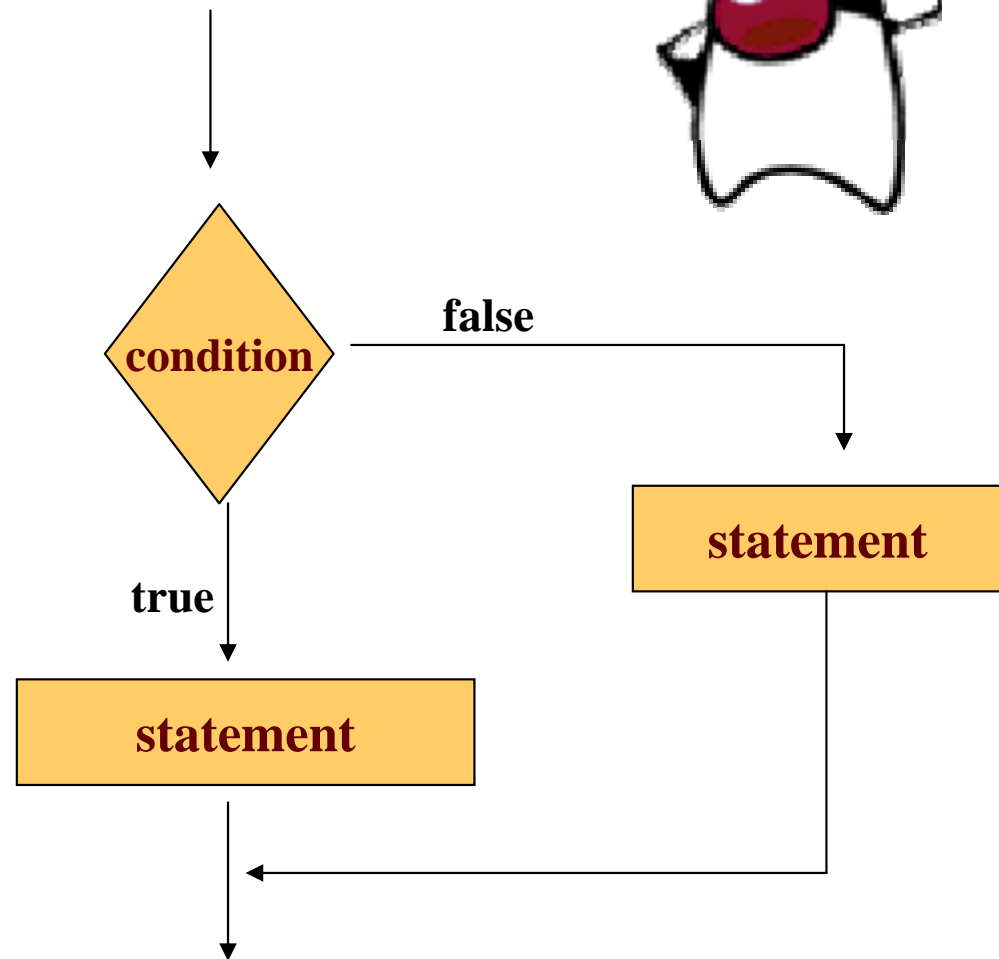
```
if (condition) {  
    statement;  
}
```



if – else Statement

Syntax:

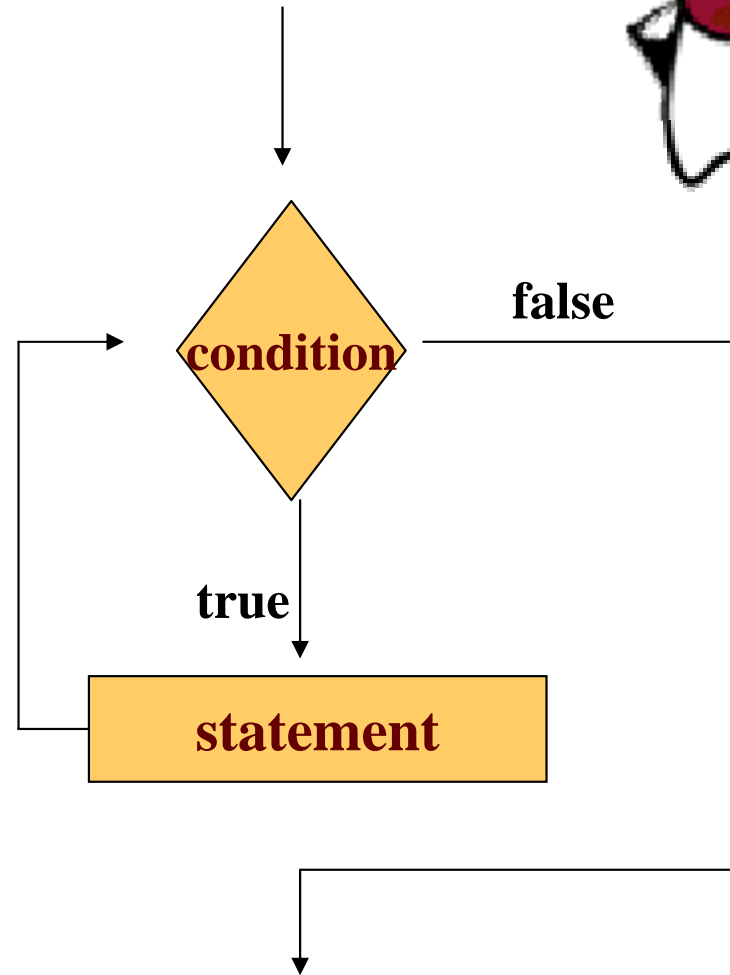
```
if (condition) {  
    statement;  
}  
else {  
    statement;  
}
```



while Statement

Syntax:

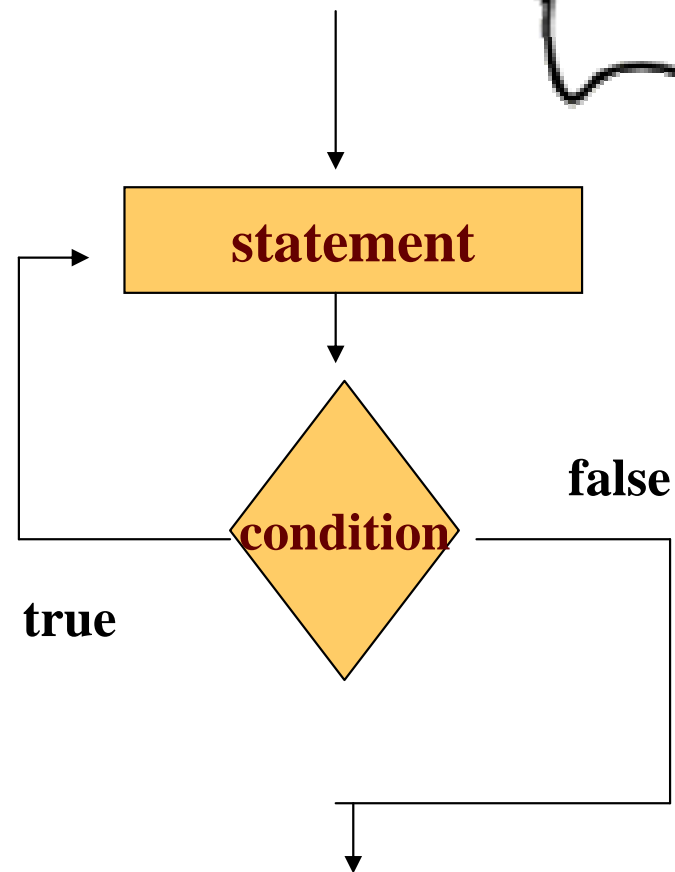
```
while (condition) {  
    statement;  
}
```



do - while Statement

Syntax:

```
do {  
    statement;  
} while (condition)
```



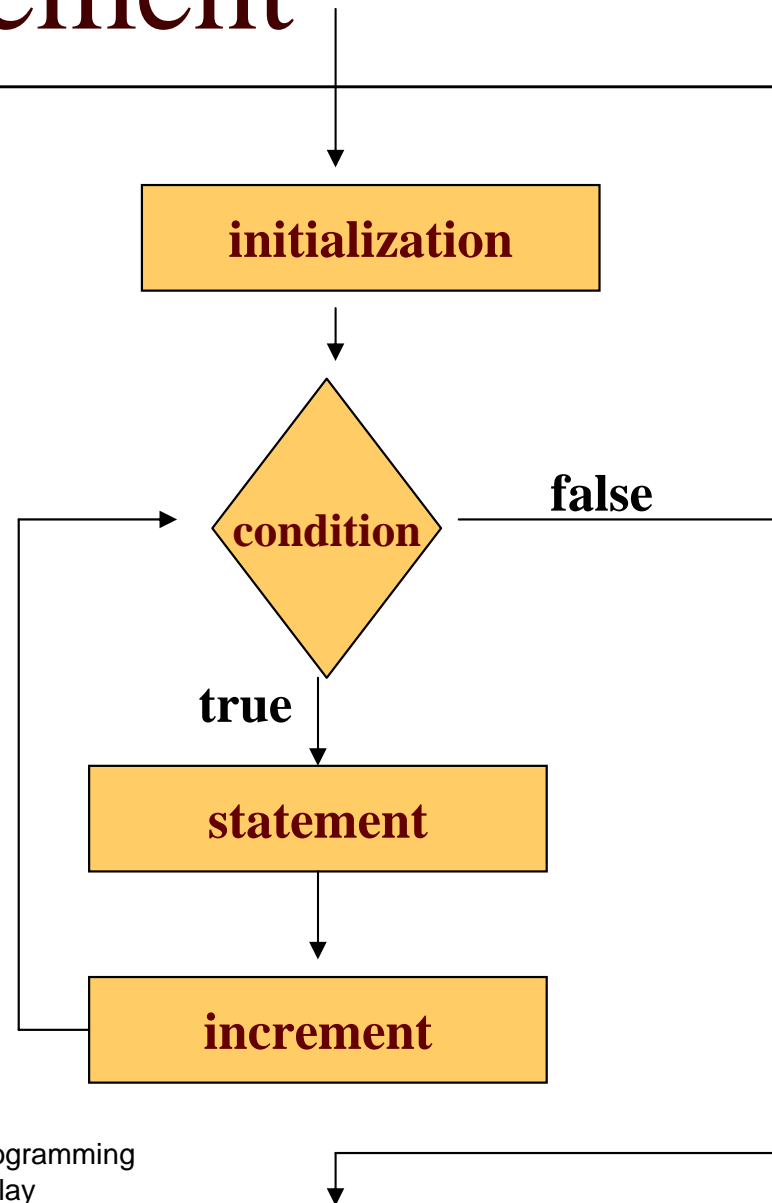
for Statement

Syntax:

```
for (initialization; condition; increment) {  
    statement;  
}
```



for Statement



switch Statement

- The `switch` statement is a choice between doing several things (usually more than two things).
- The `switch` statement evaluates an expression, then attempts to match the result to one of a series of values.
- Execution transfers to statement list associated with the first value that matches.

Switch Syntax

```
switch(exp) {  
  case value1:  
    statement1;  
    break;  
    ...  
  case valueN:  
    statementN;  
    break;  
  default:  
    defaultStatement;  
    break;  
}
```


The Break Statement

- The `break` statement is usually used to terminate the statement list of each case. This causes the control to jump to the end of the `switch` statement and continue.
- Note: if the `break` statement is omitted execution continues (“falls”) to the next case!

Example

```
switch(letter) {  
  case 'a':  
    System.out.println("The letter was a");  
    add();  
    break;  
  case 'd':  
    System.out.println("The letter was d");  
    delete();  
    break;  
  default:  
    System.out.println("Illegal input");  
    break;  
}
```

break inside a Loop

- The `break` statement, (already used within the `switch` statement), can also be used inside a loop
- When the `break` statement is executed, control jumps to the statement after the loop (the condition is not evaluated again)

continue inside a Loop

- A similar statement to the `break` is `continue` inside loops.
- When the `continue` statement is executed, control jumps to the end of the loop and the condition is evaluated (possibly entering the loop statements again).

Labeled continue and break

- In nested loops we might want to break/continue with respect to the outer loop
- As there is no goto in java – labels serves the missing feature

Labeled break

outer:

```
do {  
    statement1;  
    do {  
        statement2;  
        if ( condition ) {  
            break outer;  
        }  
        statement3;  
    } while ( test_expr );  
  
    statement4;  
  
} while ( test_expr );
```

Labeled continue

```
test:
    do {
        statement1;
        do {
            statement2;
            if ( condition ) {
                continue test;
            }
            statement3;
        } while ( test_expr );

        statement4;

    } while ( test_expr );
```

Packages and Import statements

Java API (Packages)

- ❑ Java comes with 3,000+ pre-designed components.
- ❑ The Java API is the library of classes supplied by Java.
- ❑ The classes in the Java API is separated into packages. Each package contains a set of classes that are related in some way.

The Java API Packages

`java.applet`

`java.awt`

`java.beans`

`java.io`

`java.lang`

`java.math`

`...`

`java.net`

`java.rmi`

`java.security`

`java.sql`

`java.text`

`java.util`

Documentation:

<http://java.sun.com/j2se/1.5.0/docs/api/>

The screenshot shows the Java 2 Platform API Specification website. The browser window title is "All Classes - Microsoft Internet Explorer". The address bar shows the URL <http://java.sun.com/j2se/1.4/docs/api/>. The page content includes a navigation menu with "Overview", "Package", "Class", "Use", "Tree", "Deprecated", "Index", and "Help". The main heading is "Java™ 2 Platform, Standard Edition, v 1.4.0 API Specification". Below this, it states "This document is the API specification for the Java 2 Platform, Standard Edition, version 1.4." and "See: [Description](#)".

Three callouts are present:

- List of Packages:** A callout pointing to the left sidebar, which contains a list of package names: [java.awt.image.renderab](#), [java.awt.print](#), [java.beans.BeanContext](#), [java.io](#), [java.lang](#), [java.lang.ref](#), [java.lang.reflect](#), and [java.math](#).
- List of Classes:** A callout pointing to the "All Classes" section in the left sidebar, which lists various classes such as [AbstractAction](#), [AbstractBorder](#), [AbstractButton](#), [AbstractCellEditor](#), [AbstractCollection](#), [AbstractColorChooserPanel](#), [AbstractDocument](#), [AbstractDocument.AttributeKey](#), [AbstractDocument.ContentAreaElement](#), [AbstractDocument.Element](#), [AbstractInterruptibleCharacterCache](#), [AbstractLayoutCache](#), [AbstractLayoutCache.NoCache](#), [AbstractList](#), [AbstractListModel](#), [AbstractMap](#), [AbstractMethodError](#), [AbstractPreferences](#), [AbstractSelectableChannel](#), and [AbstractSelectionKey](#).
- Details of Packages:** A callout pointing to the "Java 2 Platform Packages" table, which provides descriptions for several packages:

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.

java.lang

Choose
java.lang
from list of
Packages

List of
Classes
defined in
Package

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package java.lang

Provides classes that are fundamental to the design of the Java programming language.

See:

[Description](#)

Interface Summary

CharSequence	A <code>CharSequence</code> is a readable sequence of characters.
Cloneable	A class implements the <code>Cloneable</code> interface to indicate to the <code>Object.clone()</code> method that it is legal for that method to make a field-for-field copy of instances of that class.
Comparable	This interface imposes a total ordering on the objects of each class that implements it.
Runnable	The <code>Runnable</code> interface should be implemented by any class whose instances are intended to be executed by a thread.

Class Summary

Boolean	The <code>Boolean</code> class wraps a value of the primitive type <code>boolean</code> in an object.
Byte	The <code>Byte</code> class wraps a value of primitive type <code>byte</code> in an object.
Character	The <code>Character</code> class wraps a value of the primitive type <code>char</code> in an object.
Character.Subset	Instances of this class represent particular subsets of the Unicode character set.

String Class

The screenshot shows the Java documentation for the `String` class. At the top, there are navigation links: [Overview](#), [Package](#), **Class**, [Use Tree](#), [Deprecated](#), [Index](#), and [Help](#). On the right, it says "Java™ 2 Platform Std. Ed. v1.4.0". Below the navigation, there are links for [PREV CLASS](#), [NEXT CLASS](#), [FRAMES](#), and [NO FRAMES](#). A summary line includes [FIELD](#), [CONSTR](#), and [METHOD](#). The main content area shows the package `java.lang` and the class `String` extending `Object`. It lists implemented interfaces: `CharSequence`, `Comparable`, and `Serializable`. Below this is the class signature: `public final class String` extending `Object` and implementing `Serializable`, `Comparable`, and `CharSequence`. A paragraph explains that `String` represents character strings and that all string literals are instances of this class. Another paragraph states that strings are constant and immutable, and can be shared.

Class Hierarchy

`java.lang`
Class String
[java.lang.Object](#)
|
+--`java.lang.String`

All Implemented Interfaces:
[CharSequence](#), [Comparable](#), [Serializable](#)

`public final class String`
extends [Object](#)
implements [Serializable](#), [Comparable](#), [CharSequence](#)

The `String` class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because `String` objects are immutable they can be shared. For example:

String Methods

Method Summary

char	<code>charAt</code> (int index) Returns the character at the specified index.
int	<code>compareTo</code> (Object o) Compares this String to another Object.
int	<code>compareTo</code> (String anotherString) Compares two strings lexicographically.
int	<code>compareToIgnoreCase</code> (String str) Compares two strings lexicographically, ignoring case considerations.
String	<code>concat</code> (String str) Concatenates the specified string to the end of this string.
boolean	<code>contentEquals</code> (StringBuffer sb) Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
static String	<code>copyValueOf</code> (char[] data) Returns a String that represents the character sequence in the array specified.
static String	<code>copyValueOf</code> (char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
boolean	<code>contains</code> (String substring)

Methods List

Importing Packages

- Using a class from the Java API can be accomplished by using its fully qualified name:

```
java.util.Random random =  
    new java.util.Random();
```

- Or the class can be imported once with the import statement at the top of the file:

```
import java.util.Random;  
.  
.  
.  
Random random = new Random();
```

Importing Packages

- You can also import all the classes in a given package with a single import statement:

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

- The `java.lang` package is automatically imported into every Java program.

Arrays

Arrays

- ❑ An array is an object that can be used to store a list of values.
- ❑ All array elements are of the same type (primitive or objects).
- ❑ Arrays have fixed sizes, set when the array is created.

Array Elements

- A particular value in an array is referenced using the array name followed by the index in brackets.
- As in C, a java Array of size n is indexed from 0 to $n-1$.
- The Java interpreter will throw an exception if an array index is out of bounds.

Creating Arrays

- Array elements are initialized with their default values

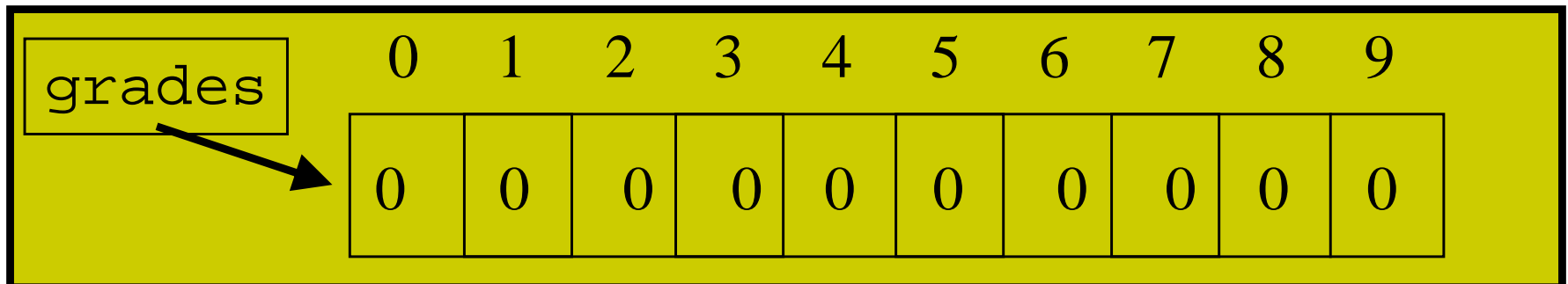
```
int[] grades = new int[10];
```

```
grades[3] = 70;
```

```
grades[7] = 87;
```

```
int i = 5;
```

```
grades[i/2] = 39;
```



Initialization list

- You can declare, construct, and initialize the array all in one statement:

```
int[] primes = {2, 3, 5, 7, 11, 13, 17, 19};
```

- This declares an array of type `int`, constructs an array of 8 slots, and assigns the designated values into the array.

```
int[][] values = {{1, 2, 5}, {4, 3, 2, 1}, {11}};
```