The Basic

Java comparing to C\C++

- Java does not have an explicit pointer type.
- Arrays in Java are class objects.
- Strings in C and C++ are arrays of characters terminated by a null character ('\0'). Strings in Java are objects.
- The memory management in Java is mostly automatic.

Java comparing to C\C++

- All Java primitive data types (char, int, long and so on) have consistent sizes and behavior across platforms and operating systems.
- Operators precedence and association are the same as in C\C++.

Java comparing to C\C++

- Although the `if`, `while`, `for` and `do..while` statements in Java are syntactically like in C\C++, in Java the condition expression must return a boolean value. In C\C++ the expressions can return an integer.

Java comparing to C\C++

- Passing over command-line arguments in Java doesn't include the application name as in C\C++.
- Java doesn’t support the ability to call a method and pass over arguments by reference.

Java comparing to C\C++

- Unlike C++, the Java programming language doesn't support multi-inheritance.
- Java doesn't allow having variables, methods and code outside the scope of a class definition.
- Unlike C++, the class type variable isn’t an object. It merely holds a reference for an object.

Simple Output commands

```
System.out.println("I have something to say …");

System.out.print("I have something to say …");

System.out.println("The value of num1 is " + num1);
```

Remarks

- C style

```
/* This is a great remark.
It is a smiley remark ….   */
```

- C++ style

```
// This is also a nice remark
```

Remarks

- JavaDoc style

```
/**
* This is a great remark.
* It is a smiley remark   */
```

The Native Data Types

- The following table summarize the native data types in Java:

Type	**Values**	**Number Of Bits**	**Default Value**
boolean	true or false	1	false
byte	integers	8	0
char	Unicode values	16	\x0000

The Native Data Types

Type	Values	Number Of Bits	Default Value
short	integers	16	0
Int	integers	32	0
long	integers	64	0
float	Real numbers	32	0.0
double	Real numbers	64	0.0

Declaring A Variable

- Declaring a variable in Java is the same as in C\C++:

```
type variableName;
```

- The following are examples for the various possible variables declarations:

```
int number;
int num,sum,total;
```

Declaring A Variable

- Like in C\C++ it is possible declaring more than one variable in the same line and it is also possible to initialize the variable in the same line.
- The following is a small example

```
int numOfStudents, numOfTeachers=22;
```

Identifiers

- Identifiers are the names we give to variables, classes and methods.
- Identifier can start with the dollar sign ($), a Unicode letter or the underscore sign (‘_’).
- The identifiers are case sensitive and they don’t have any maximum length.

Keywords

- The keywords in Java have a special meaning for the Java Compiler.
- They can be either the name of a data type or a program construct.

Expressions & Operators

- Mathematic operators:

```
+, -, *, \, %, ++, --, <<, >>, >>>, &, |, ~
```

- Logical operators:

```
!, &&, ||, ^
```

- Operators that compare between expressions:

```
<, <=, ==, !=, >, >=
```

Simple & Compound statements

- As in C\C++, Java allows writing simple statements as well as compound statements (also known as blocks).
- Each place where a simple statement can be placed, a compound statement can be placed as well.

Control Statements

- The `if` statement syntax:

```
if(boolean expression)

     statement \ compound statement
```

Control Statements

- The `if..else` statement syntax:

```
if(boolean expression)
     statement \ compound statement
else
     statement \ compound statement
```

Control Statements

- The switch statement syntax:

```
switch( expression 1 )
{
   case constant1:
            statement\s
            break;
   case constant1:
            statement\s
            break;
```

Control Statements

```
    case constant3:
            statement\s
            break;
    case constant4:
            statement\s
            break;
    `default:
            statement\s
}
```

Control Statements

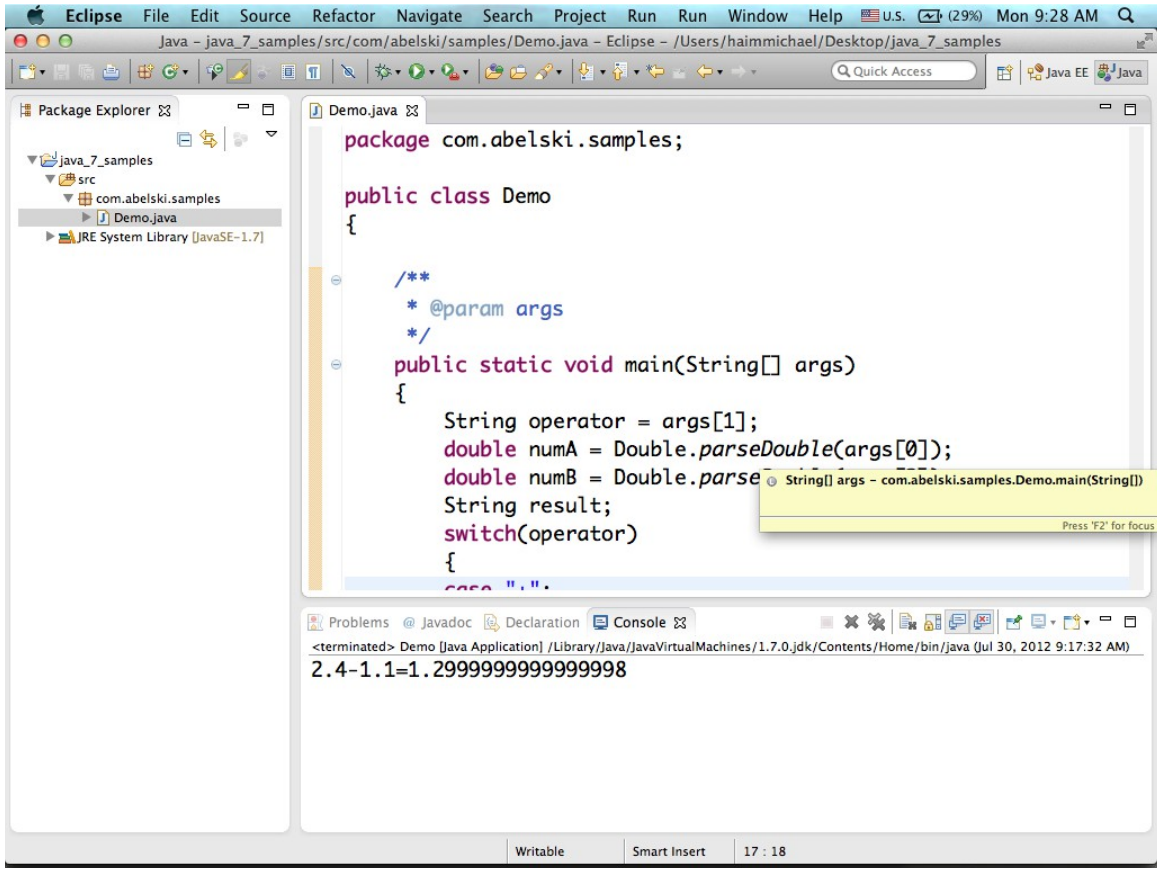
```
package com.abelski.samples;

public class Demo
{
        public static void main(String[] args)
        {
                String operator = args[1];
                double numA = Double.parseDouble(args[0]);
                double numB = Double.parseDouble(args[2]);
                String result;
                switch(operator)
                {
                case "+":
                        result = numA+"+"+numB+"="+(numA + numB);
                        break;
                case "-":
                        result = numA+"-"+numB+"="+(numA-numB);
                        break;
```

Control Statements

```
                    case "*":
                            result = numA+"*"+numB+"="+(numA*numB);
                            break;
                    case "/":
                            result = numA+"/"+numB+"="+(numA/numB);
                            break;
                    default:
                            result = "you can use one of the following "+
                                            "operators +,-,/ or *";
                            break;
                    }
                    System.out.println(result);
            }
    }
```

Control Statements


Eclipse File Edit Source Refactor Navigate Search Project Run Run Window Help
Java – java_7_samples/src/com/abelski/samples/Demo.java – Eclipse – /Users/haimmichael/Desktop/java_7_samples
Package Explorer
java_7_samples
src
com.abelski.samples
Demo.java
JRE System Library [JavaSE-1.7]
Demo.java
Problems
Javadoc
Declaration
Console
Writable
Smart Insert
17 : 18

```java
package com.abelski.samples;

public class Demo
{

    /**
     * @param args
     */
    public static void main(String[] args)
    {
        String operator = args[1];
        double numA = Double.parseDouble(args[0]);
        double numB = Double.parse
        String result;
        switch(operator)
        {
```

String[] args – com.abelski.samples.Demo.main(String[])

Press 'F2' for focus

```
<terminated> Demo [Java Application] /Library/Java/JavaVirtualMachines/1.7.0.jdk/Contents/Home/bin/java (Jul 30, 2012 9:17:32 AM)
2.4-1.1=1.2999999999999998
```

Looping Statements

- The `for` statement:

```
for(initial exp ; boolean exp ; alter exp )

     statement \ compound statement
```

Looping Statements

- The `while` statement:

```
while(boolean expression)

    statement \ compound statement
```

Looping Statements

- The `do..while` statement:

```
do
    statement \ compound statement
while(boolean exp)
```

Looping Statements

- The '`break`' and '`continue`' statements:
  - `break` [*label*]
  - `continue` [*label*]
  - `label:` *loop statement*

Looping Statements

```
public class LabelContinueExample
{
    public static void main(String args[])
    {
        outerLoop:for (int index=0; index<4; index++)
        {
            for (int counter=0; counter<12; counter++)
            {
                if (index==counter) continue outerLoop;
                System.out.println(index+”,”+ counter);
            }
        }
    }
}
```

Looping Statements

```
public class LabelBreakExample
{
    public static void main(String args[])
    {
        outerLoop:for (int index=0; index<4; index++)
        {
            for (int counter=0; counter<12; counter++)
            {
                if (index==counter) break outerLoop;
                System.out.println(index+”,”+ counter);
            }
        }
    }
}
```

Assignment Statements

- The assignment statement in Java is the same as in C\C++.
- Java supports short assignment operators as in C\C++:

  `+=, -=, *=, …`

Local Variable Scope

- Local variables are defined inside a method\block.
- Local variables are created when the method\block is executed and destroyed when the method\block ends.
- Local variables must be initialized before they are used. If a local variable isn't initialized a compile time error occurs.

Assertion

- Using Assertions we can detect errors that otherwise could be undetected and go unnoticed.
- An assertion contains a boolean expression that defines the correct state of our program at specific points in its source code.

Assertion

- A given program executes correctly if given a correct input it terminates with the correct output.

  Precondition

  Precondition is a condition the caller of the code agrees to satisfy.

  Postcondition

  Postcondition is a condition the code promises to fulfill.

  Invariant

  Condition that should always be true at a specific point of the program.

Assertion

- Writing code using preconditions, post-conditions and invariants is known as the “Design By Contract” programming model.

Assertion

- The `assert` statement can have two possible forms:

```
assert booleanExpression;
assert booleanExpression : errorMessage;
```

- The assertion facility is disabled by default. In order to enable it during runtime you should use the '`-ea`' option.

```
java -ea AssertionDemo
```

Assertion

- An assertion in Java language is a boolean expression that if evaluates as `false` we get an error message.

```
HistoryBook book = null;
book = getHistoryBook();
assert book !=null;
```

If `book` is null an `AssertionError` is thrown and all code after the assert statement won't be executed.

Assertion

```
class AssertionDemo
{
    public static void main(String args[])
    {
        assert args.length==2;
        double num1 = Double.parseDouble(args[0]);
        double num2 = Double.parseDouble(args[1]);
        double sum = num1 + num2;
        System.out.println(num1+"+"+num2+"="+sum);
    }
}
```

Assertion

```
class AssertionDemoErrorMessage
{
    public static void main(String args[])
    {
        assert args.length==2:"must send two arguments";
        double num1 = Double.parseDouble(args[0]);
        double num2 = Double.parseDouble(args[1]);
        double sum = num1 + num2;
        System.out.println(num1+"+"+num2+"="+sum);
    }
}
```

Assertion

```
class ClassInvariantDemo
{
  private static double vec[];
  public static boolean inv()
  {
    if(vec!=null && vec.length<100 && vec.length>0)
        return true;
    else
        return false;
  }
```

Assertion

```
  public static void main(String args[])
  {
     assert args.length==1:"you must set the array size";
     vec = new double[Integer.parseInt(args[0])];
     assert inv():"class invariant is not true";
  }
}
```

Annotations

- Many APIs require an additional code (e.g. XML deployment and configuration files, additional standard classes etc..).
- The Java platform allows adding annotations within the code in order to indirectly instruct the platform to create a required XML file, define additional class or work in a specific way.

Annotations

❖ There are many sorts of annotations we can add into the code. The following are few sample possible annotations that have always been supported by the Java platform:

+ Adding the “@deprecated” javadoc tag will indicate the marked method should no longer be used.

+ Adding the “transient” modifier indicates the marked field should be ignored during the serialization process.

Annotations

- Since Java 5.0 the Java platform has a general purpose annotation mechanism (AKA “metadata facility”) that enables us to define and use our own defined annotation types.

Annotations

- The syntax used when defining an annotation is very similar to the one we use when defining an interface.
- The '@' sign precedes the “interface” keyword.

```
public @interface MyAnnotation
{
    int id();
    String info();
}
```

Annotations

```
...
@MyAnnotation
{
    id = 123123;
    info = “Do Something Good!”;
}
...
```

Number Systems

- We can easily express integral numbers using the binary, the octal and hexadecimal number systems.

```
int numA = 0b10011101; //binary
int numB = 03425; //octal
int numC = 0xE12F; //hexadecimal
```

- The support for the binary number system was added in Java 7.

Number Systems

- We can easily express integral numbers using the binary, the octal and hexadecimal number systems.

```
int numA = 0b10011101; //binary
int numB = 03425; //octal
int numC = 0xE12F; //hexadecimal
```

- The support for the binary number system was added in Java 7.

Underscores in Numeric Literals

- As of Java 7, we can improve the readability of our code by adding underscores in between digits in numerical literals.

```
double num = 1_424_234.532;
```