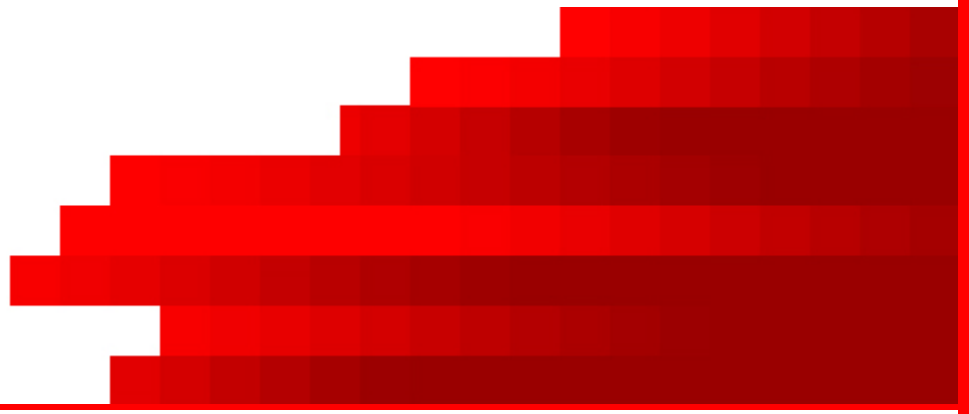
Core Java



HSBC Technology and Services

HSBC
The world's local bank

Agenda

- History
- How programs are Platform-dependent?
- Platform Independence of Java
- Byte Code
- Three essential parts of the Java platform
  - The Java Virtual Machine.
  - The Java APIs.
  - The Java language.
- Java source code, generation of **`.class`** files

Agenda

- Java Features
  - strong typed.
  - Robust etc
- Strings are immutable
- Arrays
  - A**`rrayIndexOutOfBoundsException.`**
- Operators
- Control flow

Origins and History

- In the early 90's, a development team in Sun Microsystems, led by James Gosling, had an ambitious vision:
  - What they had in mind was the creation of a new programming platform.
  - This platform would enable the development and execution of software applications.
  - Their vision's special feature was that it allowed for a code that was written and complied of one platform to be run on its peer.
    - The peer can be installed in a completely different operating environment.

Smart Appliances

- The motivation behind their vision was to create a platform that can be also embedded inside non-desktop computers.
  - Their target was smart appliances (e.g., VCRs, Personal Digital Assistants, pagers, cameras, printers) that are connected to a network.
    - While running the new platform they may all share services.
- Even in those early days of the Internet, Sun people spotted the potential of the World Wide Web.
  - The platform architecture was aimed at creating a network-oriented software technology.

The Java Revolution Begins

- Sun introduces the first universal software platform, designed from the ground up for the Internet and corporate intranets. Java technology enables developers to write applications only once to run on any computer (1).

(1) Sun's formal history - www.sun.com/aboutsun/coinfo/history.html

The Wheel Keeps Rolling

- Since its introduction in May 1995, the **Java** platform has been adopted more quickly across the industry than any other new technology in computing history.
  - Only three years later, in 1998, about a million programmers downloaded the Java Development Kit from Sun's site.
- All major computing platform vendors have signed up to integrate Java technology as a core component of their products.

Java's Key Features

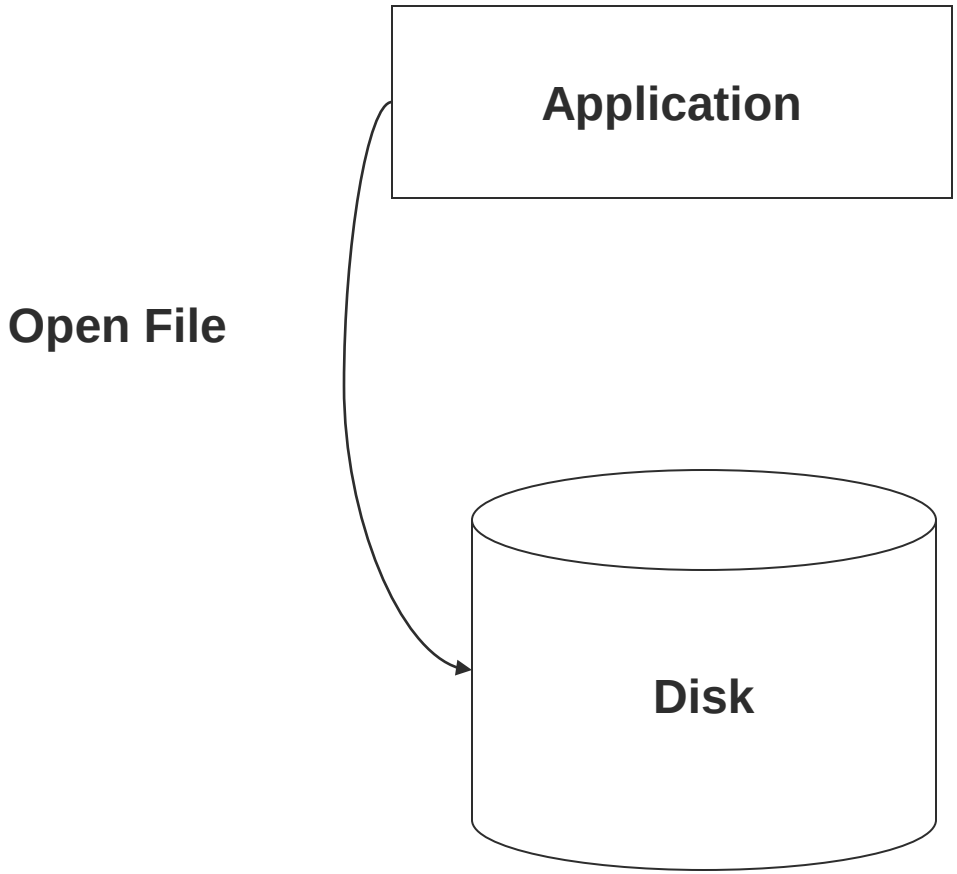
- This new network-oriented language architecture consists of three outstanding benefits:
  - **Robustness** - Java is more robust than other known technologies:
    - An array whose declared bounds were exceeded may cause a lot of memory damage in C, but in Java it will just throw an exception.
  - **Automatic memory management** – Thanks to the **garbage collection** you don't have to mess around with mermory management.
  - **Platform-independency** - The ability to code once and execute on multiple platforms.
    - Subject to the installment of the Java platform.

Platform-Independence

- In the next slides, I will take you step-by-step inside the world of platform-independency.
  - I will present you with a simple task performed both in C and in Java.
  - Then, I will take a first glance at the **byte code** and **Virtual Machine** concepts.

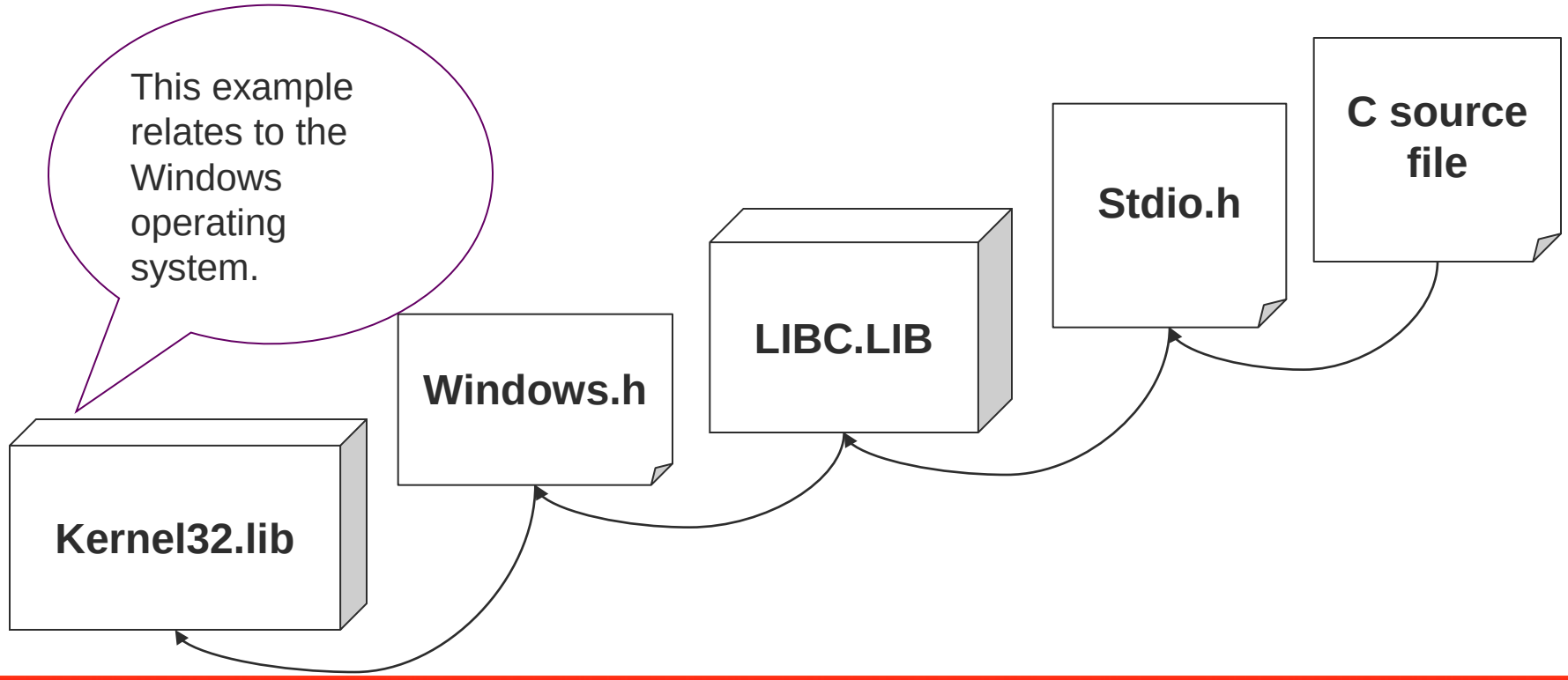
Opening a File

Let's assume you are coding an application for opening a file located on the local hard drive:



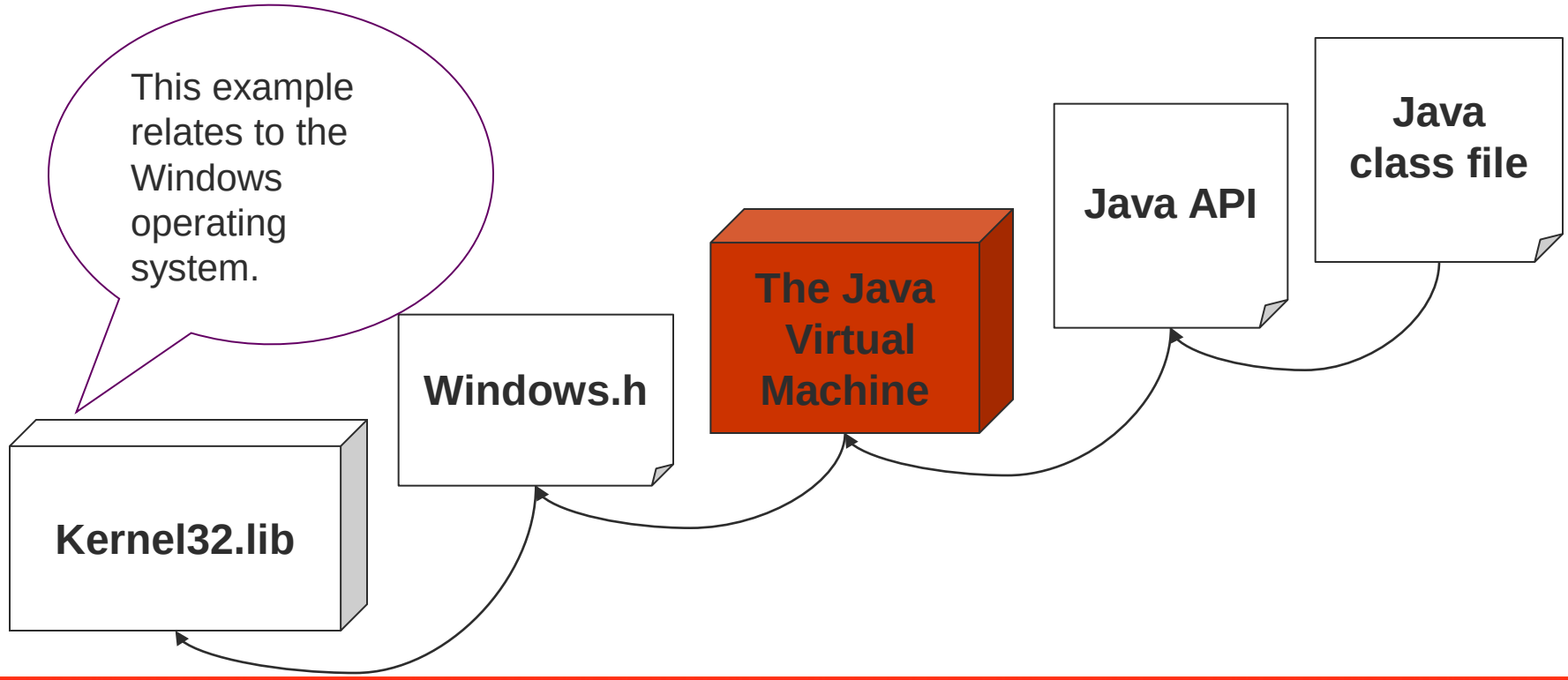
Scenario #1 – Using C

```
#include <stdio.h>
Void main()
{
        FILE *fRead = fopen("c:\\MyFile.txt");
        fscanf…
}
```



Scenario #2 – Using Java

```
Public static void main(String args[])
{
        FileInputStream  in = new FileInputStream();
        in.read() . . . . .
}
```



C vs. Java - cont'd

C

```
1.fOpen(...)
2.-> Declared in stdio.h
3.--> Implemented in LIBC.LIB
4.---> Uses Win32 API CreateFile
5.----> Declared in Windows.h
6.-----> Implemented in Windows.lib
```

The connection to the O.S functions is already known at compile time.

Java

```
1.createNewFile( )
2.-> Declared in Class File
3.--> Implemented in java API
```

At compile time we are only familiar with the Java platform API methods.

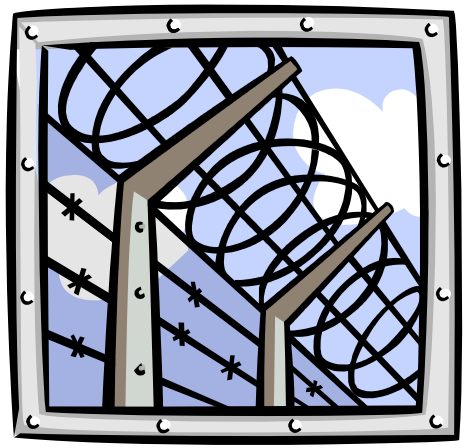
```
4.---> Uses Win32 API CreateFile
5.----> Declared in Windows.h
6.-----> Implemented in Windows.lib
```

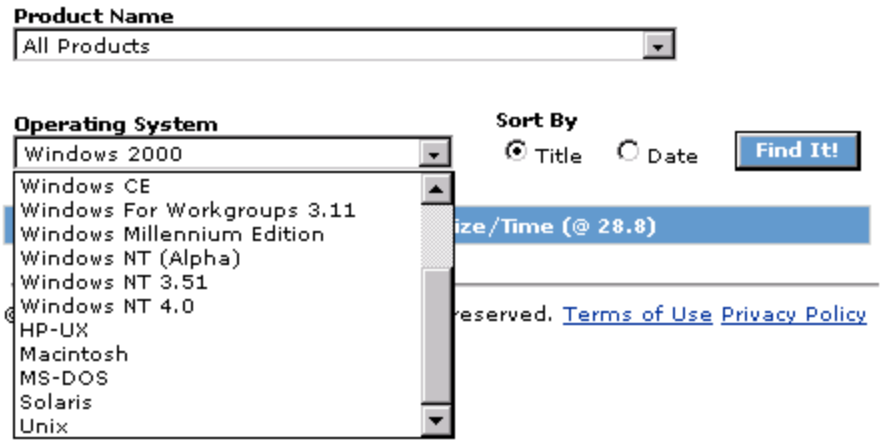
The Java Virtual Machine (JVM) knows how to do the mapping between the API methods to O.S. functions at Run time.

The Platform Jail

- A Compiled C application can be execute only on the same platform that it was compiled and linked on.
  - In some cases, the program might run of a platform that is a member of the same family (Win ME / 2000).

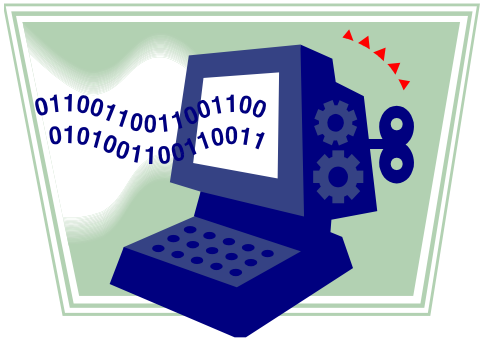
- For example, an application built on the Linux Operating System will not run on Windows.
  - Each system may define a different structure for its executable files using alternative native resources.





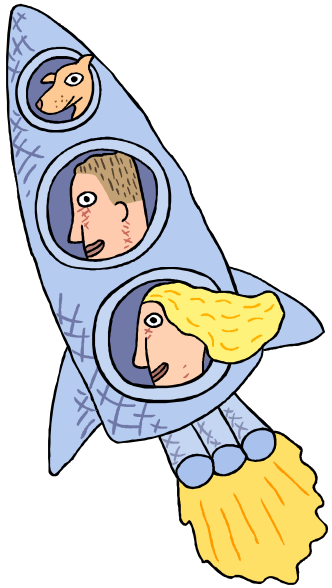
Byte Code

- In Java, everything is aimed at enhancing network mobility.
  - The file's format is no exception.
  - The `.java` files are complied into `.class` files.
  - The latter contain byte code.
- The name byte code reflects the fact that each instruction occupies only one byte.
  - As always, certain minor exceptions might be made.
- The `.class` files contain special binary data:
  - A set of instructions for the Java Virtual Machine.
  - The byte code is not platform specific.
  - The VM implementation might certainly be.



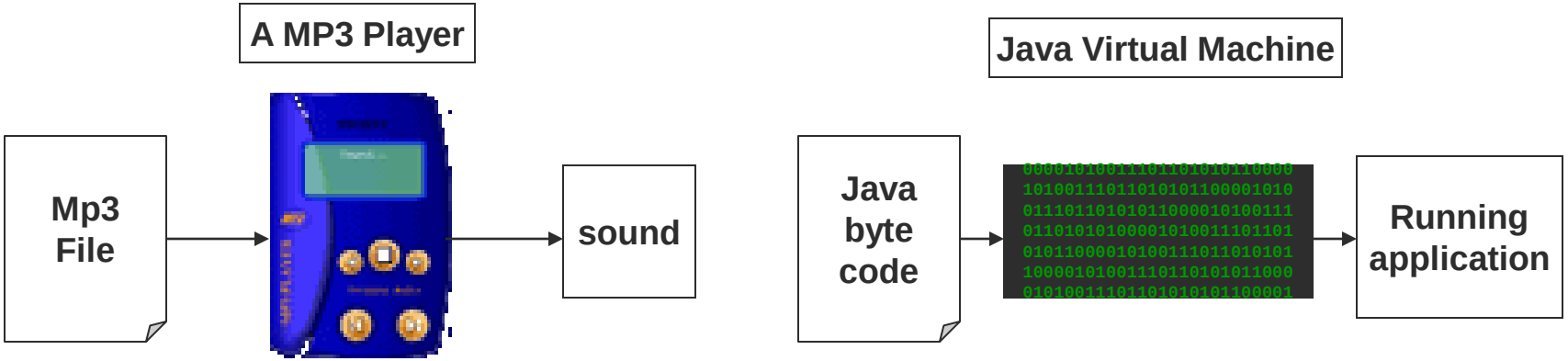
Java Virtual Machine (JVM)

- The virtual part of the name implies that this is really an abstract concept, an idea of a computer.
- To run a concrete Java application you must install an implementation of that abstract computer on your actual machine.
  - The specification allows great freedom for an implementation writer for usage of software and hardware.
- Each time you launch the Java command you create a new run-time instance of the JVM implementation.
  - Every application runs inside its own instance.
  - Running two applications at once means launching two separate instances.



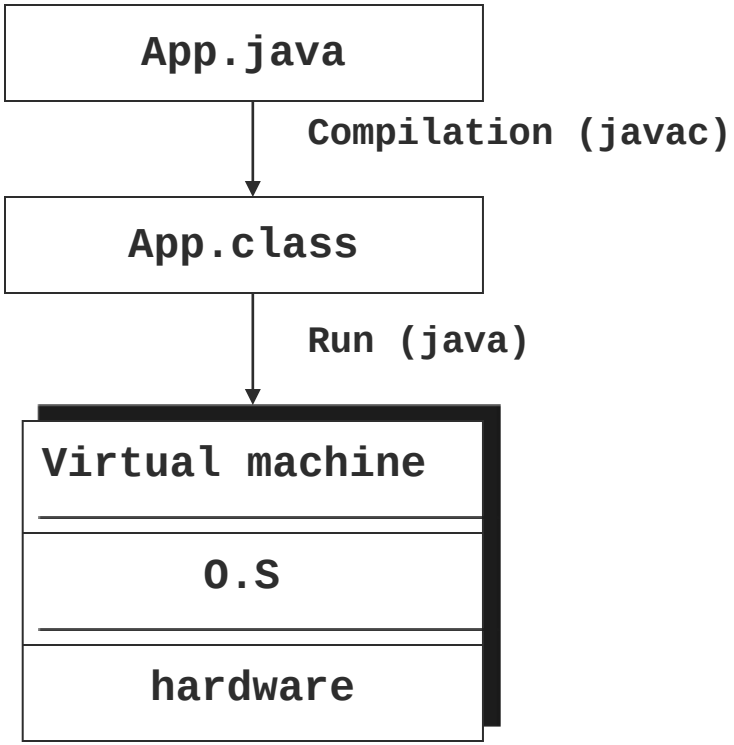
The JVM

- The JVM you will install on your Window's operated PC is not the same as the implementation for a Solaris computer.
  - What's similar between the two implementations is that both of them may receive an identical .**`class`** file and execute the same functional application.
  - The JVM interprets the Java byte code program into calls to native functions of the operating system.



Porting Java

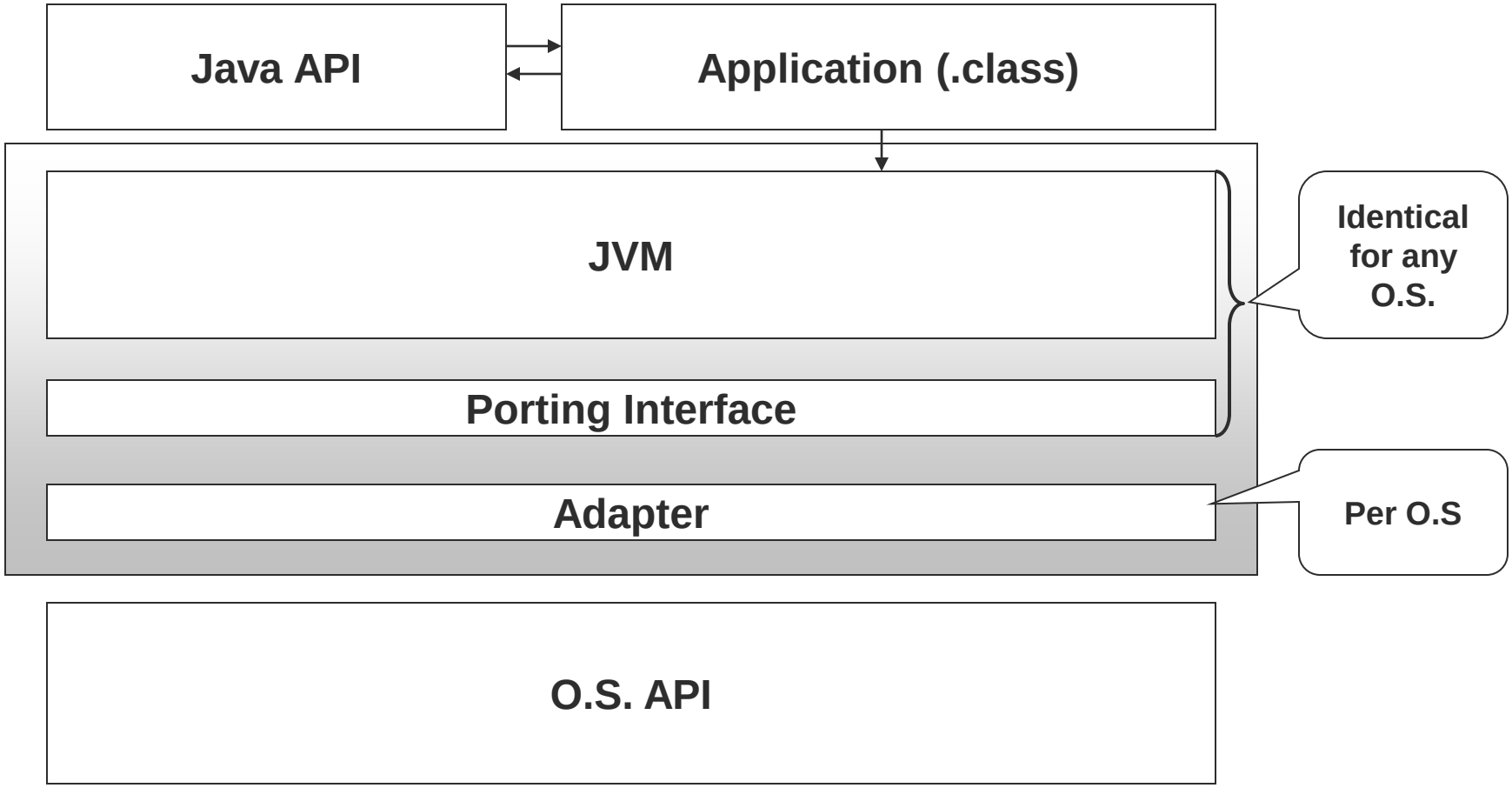
- Isolating the dependency between the application to the underlying operating system makes it possible to run the same byte code file in any platform (O.S + computer hardware) on which the Java platform is installed.



The Java Platform

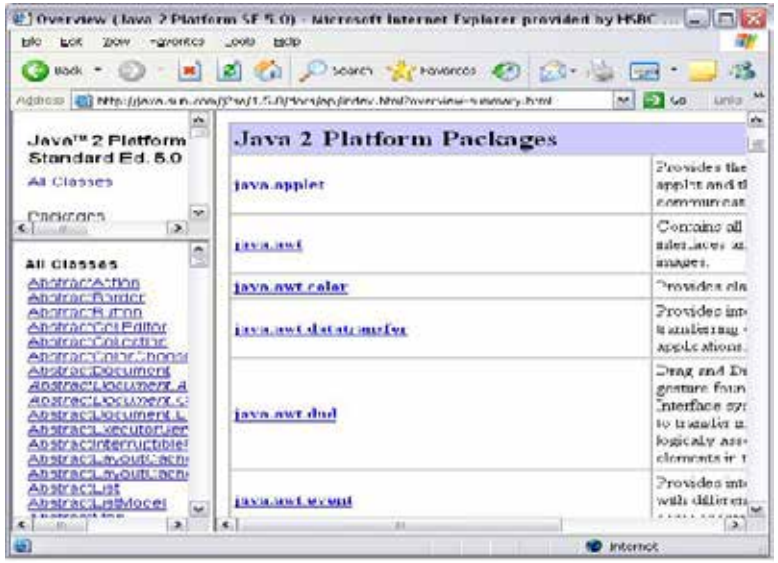
- The Java platform contains three highly related components:
  - The Java Virtual Machine (JVM).
  - The Java programming language.
  - Java Application Programming Interface (API).
- The remaining of the chapter will focus on these subjects.

The Java Platform Architecture



The Java API

- API is an acronym for: **A**pplication **P**rogram **I**nterface.
  - It is a set of essential interfaces and classes in key areas such as:
    - Communicating with the operating system.
    - Connection to a Database.
    - Resource management.
  - A developer can use the supplied APIs in the application and focus on the business logic instead.



Summary

- The Java platform consists of three essential parts:
  - The Java Virtual Machine.
  - The Java APIs.
  - The Java language.

- When compiling Java source code, the Java compiler generates .**`class`** files containing Java Virtual Machine commands, also known as byte code.

Summary

- The Java Virtual Machine interprets the **`.class`** files and converts them to specific (native) machine instructions.

Java Language

- When a Virtual Machine instance starts running, it does so by invoking a **`main()`** method.
  - Any class containing a **`main()`** method may function as the application starting point.

The `HelloWorld` class should be located inside `HelloWorld.java` file

The `main()` method should be `public` and `static`, return `void` and accept an array of type `String`

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

        {
        System.out.println("Hello world!");
        }
}
```

Printing to the standard output

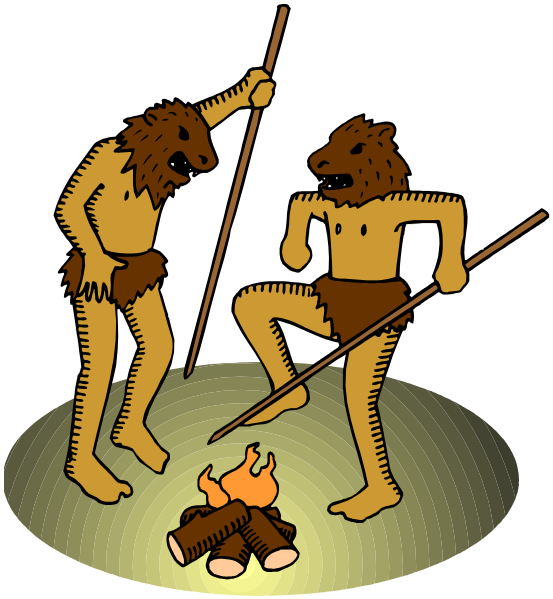
Compiling and Running

- For compiling a .java source code, we use the **javac** compiler (don't forget to indicate the .java extension).
  - `javac HelloWorld.java`

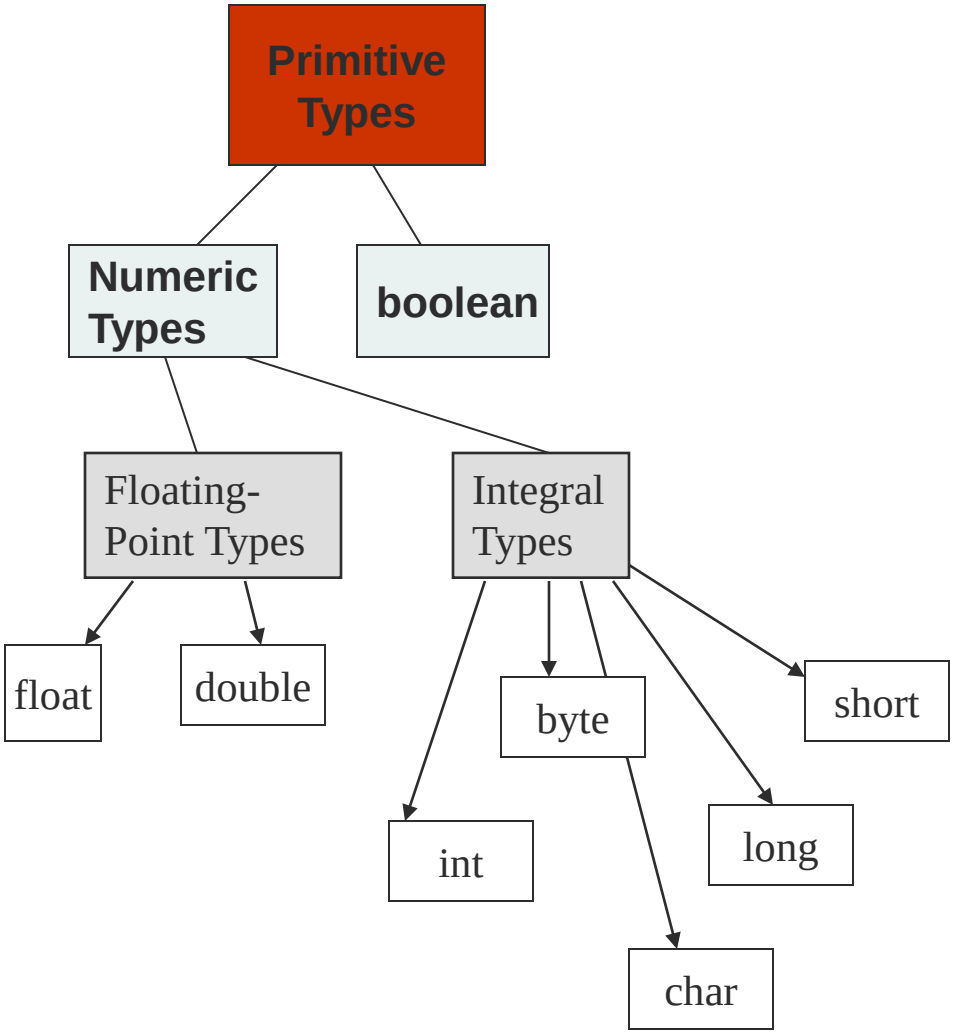
- Once the compilation is completed successfully, the compiled bytecode (.class file) can be executed using the **java** JVM (this time no extension is needed):
  - `java HelloWorld`

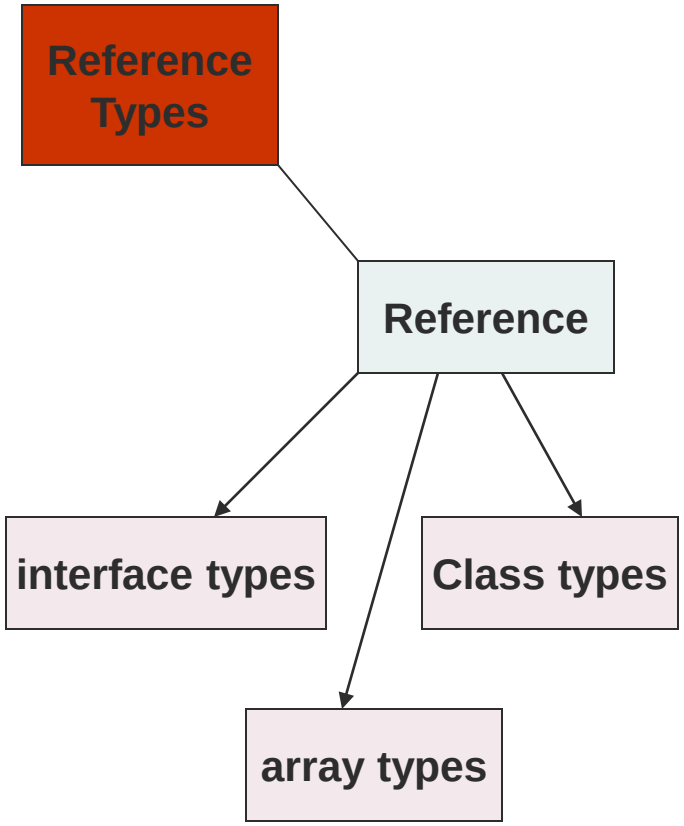
Data Types

- There are two data types in Java:
  - **Primitive types** – hold primitive values.
    - Their range is independent of the underlying host platform and is identical at all JVMs.
  - **Reference types** – hold reference values.
    - References to dynamically created objects.



Data Types


Primitive Types
Numeric Types
boolean
Floating-Point Types
Integral Types
float
double
byte
short
int
long
char


Reference Types
Reference
interface types
Class types
array types

Primitives - Numeric types

- Integral types:

`byte`	8 bit signed integer
`short`	16 bit signed integer
`int`	32 bit signed integer
`long`	64 bit signed integer
`char`	16 bit unsigned Unicode character

- Floating-point types:

`float`	32 bit single-precision float
`double`	64 bit double-precision float

Primitives - Boolean type

- Unlike all other value types, the JVM does not defines a Boolean type.

`boolean`	true or false

Primitive Variables

- A **variable** is a representation of a place inside the computer's memory. That piece of memory is associated to a data type and can be assigned a value.

```
1  int zebrasCount;
2  float      flyingSpeed;
3  int        i, j;  // declaring 2 int variables
4  long       snake;
```

- It is widely accepted to use a lower case letter as the first character of a variable name.
  - It is also common to start the names of variables which are class members with the sequence "**m**_".

Primitive Variables - cont'd

- Assignment: setting a value to a variable

- Initialization: creating a variable with a primary value

```
1. int i1;
2. i1 = 88;           // assignment
3. int   i2 = 128;    // initialization
4. char  ch = 'A';    // initialization
5. double x = 77.88; // initialization
6. x = i1;            // assignment
7. float f1 = 12.1F;
8. long l1 = 900000000000L;
```

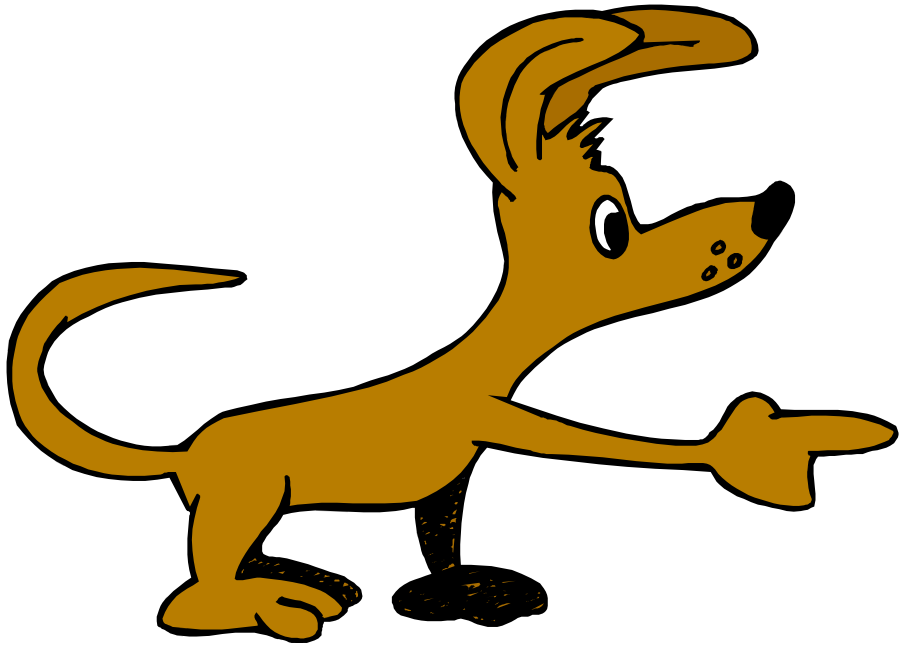
Casting

- In case an assignment may result in a lost of information, you'll need to use casting explicitly:

```
1. long lon1 = 20000L;
2. int i1 = lon1;                    // Compile error
3. i1 = (int)lon1;                   // Ok – explicit casting
4. float f1 = (float)12.5;  // OK – explicit casting
```

Reference Types

- Class types
  - References to class instances (Objects).
- Interface types
  - Reference to class instances (Objects) that implement an interface.
- Array types
  - References to arrays.



Class Types - Strings

- Class **`String`** instances are used to represent character sequences.
  - The class contains methods for examining individual characters of the sequence, comparing and searching, extracting sub-strings, and for creating a copy of the string.
- The strings are composed of Unicode characters.
  - Unicode is an international specification holding the characters of all known languages.
- It is important to notice that not all operating systems support Unicode. JVM running on such OS will not represent strings as Unicode.

- **Windows 98 and Windows 95 does not support Unicode.**
- **Windows 2000 / NT and Windows CE support Unicode.**
- **For more information on Unicode visit www.unicode.org**

String Examples

- System.out.println(“Have a nice day”);
- String string = new String(“string”);
- String string2 = “string2”;
- System.out.println(string + “ #1”);
- String sub1 = “immutable”.substring(2);
- String sub2 = string.substring(0, 2);

Strings are Immutable

- It is impossible to change the value of a String instance directly.
  - In order to perform string manipulation StringBuffer comes to the rescue.
    - **`StringBuffer`** is just like String, except for it being mutable – it can be changed.
- The following code:

```
String str = “the”;
String myString = str + “ hitchhiker” + “ guide”;
```

- Is compiled to:

```
myString = new StringBuffer().append(str).
append(“ hitchhiker”).append(“ guide”).toString();
```

Comparing Strings

- To compare 2 strings (as well as StringBuffers) use their `equals()` method.

```
1. String a="Shuky";
2. String b="Shuky";
3. if (a.equals(b))
4.     System.out.println("Strings are equal");
```

Primitive Types Arrays

- Declaring a reference to an array:

```
int[]  iArray;        // what is the value of iArray?
```

- Creating the array:

```
iArray = new int[50]; // what is the value of iArray[0]?
```

- Putting values into the array:

```
iArray[0] = 5;
iArray[1] = 55;
```

- Declaring and creating an array:

```
byte[] bArray = new byte[100];
```

- Creating an array and initializing it:

```
double[] dArray = { 1.1, 2.2, 3.445 };
```

Arrays - cont'd

1

Declaration:

```
int[]  iArray;
int    iArray[];
```

2

Allocation:

```
iArray = new int[20]
```

3

Assignment

```
iArray[3] = 3;
```

Declaration and Allocation:

```
int iArray[] = new int[20];
int[] iArray = new int[20];
```

Declaration Allocation and Assignment

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

Arrays - cont'd

- The following statement: **`int[] iArray = new int[50]`** sets all elements to 0 ("zero").
- Element indexes are 0..49
- **`iArray.length`** returns the number of elements in **`iArray`** array.
- Accessing element out of the array bounds such as:
  `iArray[50] = 10;`
  will throw the exception: **`ArrayIndexOutOfBoundsException.`**
- Once created, array has a fixed size.

Java Arrays vs. C/C++ Arrays

- In C, we could write the following piece of code:
  - `int arr[20];`
- This will allocate a sequence of 20 int variables, that may be accessed through the pointer '**`arr`**'.
- This is **not** a legal Java statement.
- Java arrays are not primitive types, therefore they must be instantiated using the keyword **`new`**.
- For example:
  - `int arr[] = new int[20];`

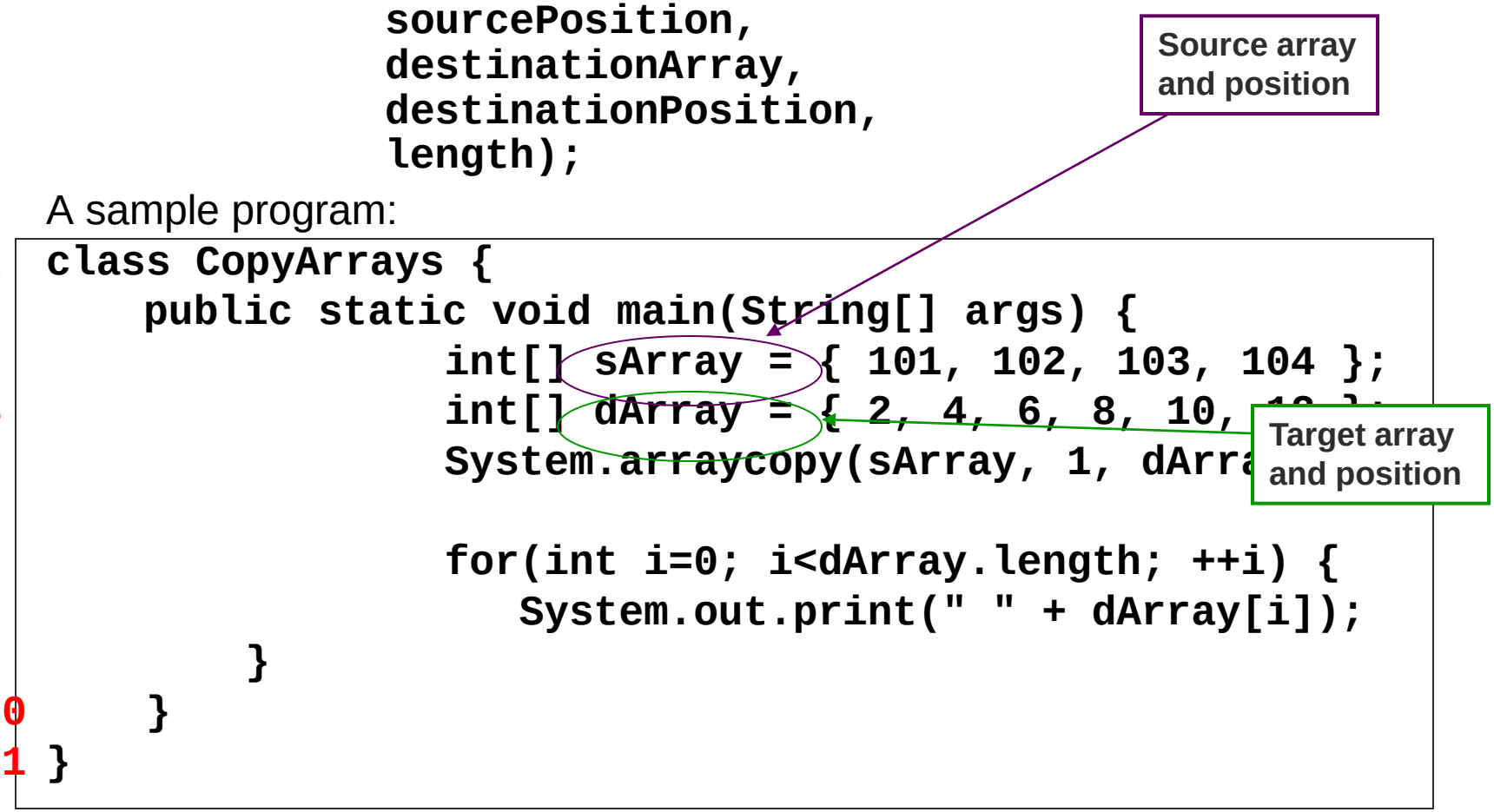
Copying an Array

- There is a special method for copying an array.

```
System.arraycopy(sourceArray,
                 sourcePosition,
                 destinationArray,
                 destinationPosition,
                 length);
```

- A sample program:

```
1  class CopyArrays {
2      public static void main(String[] args) {
3                  int[] sArray = { 101, 102, 103, 104 };
4                  int[] dArray = { 2, 4, 6, 8, 10, 12 };
5                  System.arraycopy(sArray, 1, dArray, 2, 2);
6
7                  for(int i=0; i<dArray.length; ++i) {
8                     System.out.print(" " + dArray[i]);
9              }
10     }
11 }
```



A Simple Program with Arrays

```
1   class SimpleProgramWithArrays {
2           public static void main(String[] args) {
3         // array of int
4         int[] nArr = new int[5];
5         nArr[0] = 12;
6         nArr[1] = 13;
7         for(int i=0; i<nArr.length; ++i) {
8            System.out.println(nArr[i]);
9         }
10
11        // array of strings
12        String[] sArr = new String[2];
13        sArr[0] = "Zebra";
14        sArr[1] = "Unicorn";
15        for(int i=0; i<sArr.length; i++) {
16            System.out.println(sArr[i]);
17        }
18    }
19  }
```

Java Arithmetic Operators

Operator	Function	Example
+	Addition.	5 + 9
-	Subtraction.	6 - 3
*****	Multiplication.	2 * 4
/	Division.	10 / 2
%	Modulo.	53 % 10
+=	Incrementing the LHS value by the RHS value	i += 5
-=	Subtraction of the LHS variable by the RHS value	i -= 8
/=	Division of the LHS variable by the RHS value	i /= 4
***=**	Multiplication of the LHS variable by the RHS value,	i *= 2
++	Increment the variable by 1. Increment and return value (pre-increment) Return value and then increment (post-increment)	 ++i i++
--	Decrement the variable by 1. Pre decrement Post decrement	 --i i--

Operators

} Here are some examples for using operators in Java:

```
1 int  zu = 10, lu = 20; // initialization
2 int  x = zu + lu;
3 x += 20;    // same as: x = x + 20
4 ++x;        // same as: x = x + 1
5 long  y = 2 * ++lu; // lu is 21, y is 42
```

} Java operators are almost the same as in C and C++.

Java Arithmetic Operators - cont'd

- The result type of an integer operations is an integer:

9/2 gives 4 (not 4.5)

Comparison Operators

Operator	Function	Example
==	Equals.	a == b
!=	Not equal.	a != b
<	Less than.	a < b
>	Greater than.	a > b
<=	Less than or equal to.	a <= b
>=	Greater than or equal to.	a >= b

Logical Operators

Operator	Function	Example
&&	Logical AND.	if(a > 0 && a < 10)
||	Logical OR.	if(a == 4 || a == 6)
!	Negate.	if(!(a == 4 || a == 6))

Hierarchy of Operators

- The operators hierarchy:

[ ] . ( ) (method call)	**left to right**
! ~ ++ -- + (unary) - (unary) ( ) (cast) new	**right to left**
*** / %**	**left to right**
+ -	**left to right**
<< >> >>>	**left to right**
< <= > >= instanceof	**left to right**
== !=	**left to right**
&	**left to right**
^	**left to right**
|	**left to right**
&&	**left to right**
||	**left to right**
? :	**left to right**
= += -= *= /= %= &= |= ^= <<= >>= >>>=	**right to left**

Java Conditional Statements

- In the C programming language, we make use of integer values in Boolean conditions:
  - Zero is equivalent to **false.**
  - Positive and negative value are equivalent to **true.**

```
1.  if(-5)
2.  if(5)
3.  while(1)
4.  if(0)
```

true

false

- In Java things are ~~a little different~~: the Boolean condition must test a Boolean variable, and not numeric values.

```
1.  boolean b=true;
    if(b) . . .
2.  while(true)
```

Control Flow: `if/else`

```
1. if(a > b) {
2.   System.out.println(a + " is bigger");
3. }
4.
5.
6.  if(a > b) {
7.   System.out.println(a + " is bigger");
8. } else {
9.   System.out.println(a + " is smaller");
10.}
11.
12.if(numOfDogs > numOfCats) {
13.    sound = "haw haw";
14.} else if (numOfDogs < numOfCats){
15.    sound = "miau miau";
16.} else {
17.    sound = "haw miau";
18.}
```

Control Flow: Loops

- There are three kinds of loops in Java: **`while`** loop, **`for`** loop, **`do`** loop

  A simple **`while`** loop.

```
1 int a = 1, sum = 0;
2 while(a < 10) {
3   sum += a;
4   a *= 2;
5 }
6 // how much is a ? how much is sum ?
```

A simple **`for`** loop.

```
1 for(int i=0; i<10; ++i) {
2   System.out.println(i * i);
3 }
```

Control Flow: Loops – cont'd

– A simple **do** loop.

```
1     char cArr[] = {'a','b','c','9','d'};
2     int i = 0;
3     do
4     {
5        System.out.println(cArr[i]);
6     }  while(cArr[++i] != '9');
```

Control Flow: switch

```
1  int a = getValueFromSomewhere();
2  switch(a) {
3    case 0:
4    case 1:
5      ++c1;
6      break;
7
8    case 2:
9      ++c2;
10     break;
11
12   case 3:
13     ++c3;
14     break;
15
16   default:
17     System.out.println("a is: " + a);
18     break;
19 }
```

Control Flow: break

```
1  int[] a1 = {2, 3, 6, 8, 10, 12, 14};
2  int[] a2 = {1, 50, 40, 14, 16, 40};
3  int i=0,j=0;
4
5  looop:
6  for(i=0;i<a1.length;++i)
7  {
8    for(j=0;j<a2.length;++j)
9     {
10     if(a1[i] == a2[j])
11     {
12        System.out.println("i= " +i+ " j= " +j);
13       System.out.println("equal num --> " +a1[i]);
14        break looop;
15      }
16    }
17 }
```

Conversions - Wrapper Classes

- The **`java.lang`** package includes wrapper classes for all the language's primitive types.
  - A wrapper class wraps a value of the primitive type in an object.
  - The class provides several **static** methods for manipulations and conversion between primitive types.
- The primitive type **`int`** has a corresponding class **`Integer`**.
  - One of Integer's methods is:
    `public static int parseInt(String s)`

```
1   // String concatenation:
2   String num = "123";
3   System.out.println(num+4);
4   // Adding numbers:
5   int iNum = Integer.parseInt(num);
6   System.out.println(iNum+4);
```

Exercises

1. Write a short HelloUser application.
   - Your application should receive two arguments:
     - The user's name.
     - The user's age.
   - The application output should be:
     - Hello "user name", in ten years you are going to be X.
2. Create a new class that receives a list of numbers, displays them delimited with commas and indicates the smallest value.

   For example, the prompt `java FindMin 6 10 3 20 12` Will produce:
   `6,10,3,20,12,  min=3`

Summary

- The Java platform includes 3 parts:
  - The Java language.
  - The Java API.
  - The Java Virtual Machine.
- Java application can be coded and complied on any computer running JVM.
  - The files can then be executed on any other computer and operating system.

Summary

- Java takes care of its programmers:
  - The language is strong typed.
  - It is very robust.
  - It is almost impossible to tamper with the computer's memory, the do[illegible]s is much more productive.
- Data types in Java
  - Primitive types
  - Reference types
- Casting
- Strings are immutable
- Arrays
  - A**`rrayIndexOutOfBoundsException.`**
- Operators
- Control flow

About HSBC Technology and Services

HSBC Technology and Services (HTS) is a pivotal part of the Group and seamlessly integrates technology platforms and operations with an aim to re-define customer experience and drive down unit cost of production. Its solutions connect people, devices and networks across the globe and combine domain expertise, process skills and technology to deliver unparalleled business value, thereby enabling HSBC to stay ahead of competition by addressing market changes quickly and developing profitable customer relationships.

Presenter's Contact Details:
Name: Deepak Ratnani
Role: Team Leader
Direct: + 91-20 -66423608
Email: DeepakRatnani@hsbc.co.in