Why Java?

- Object–Oriented, many packages available
- Interpreted: "byte–code", no linking
- Portable: mostly architecture–independent
- Distributed: sockets, RMI, CORBA
- Simple: no header files, no pointers
- Robust: strongly typed, garbage collection
- Secure: "sandbox" applets, signatures
- Threaded: lightweight processes, easy to code

"Hello World" – Program

```
/* Simple "Hello!" application */

public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello, World");
  }
}
```

Java vs. C – Major Differences

- No headers. No pointers. No macros. No `structs`. No `union`. No `#define`. No `#include`. No `#ifdef`
- No global variables (but public class variables)
- Packages and Imports instead of Includes
- Objects are always passed by reference, no dereferencing!!!
- Garbage collection ––> no `malloc()/free()`
- Exception handling

Passing by Reference: What does this code do?

```
public void swap (Object a, Object b) {
  Object temp = a;
  a = b;
  b = temp;
}
```

```
import java.util.*;
class Refs {
    public static void main (String[] s) {
        Vector v = new Vector();
        v.addElement(new Integer(3));
        v.addElement(new Integer(4));
        Vector v2 = v;
        System.out.println(v2);
        v.setElementAt(new Integer(100),0);
        System.out.println(v2);
    }
}
```

Data Types

- Primitive Types

Type	Size (bits)
boolean	**1**
char	**16**
byte	**8**
short	**16**
int	**32**
long	**64**
float	**32**
double	**64**
array	**–**

- Object Types

Type	Related
Boolean	
Character	**String**
Byte	
Short	
Integer	
Long	
Float	
Double	
–	**Vector**

Exceptions

- `throws`–Clause. Throwing an exception is signalling an exceptional condition.
- `try/catch`–Statements. Catching an exception is to handle it.
- Pre–Defined Exceptions and User–Defined Exceptions (classes).

Applets vs. Applications

- Applets extend `applet`–Class
  - can be displayed in browsers
  - sandbox principle —–> security issues
- Applications are standalone (command line)
  - started with `java filename` on command line
  - no security restrictions
  - may write GUI using the `java.awt.(J)Frame` and `java.awt.Window` classes

Java Swing (JFC)

- most classes start with a `J` (`JFrame, JApplet, JButton, JPane`...)
- new event model: listeners, callback functions
- separate library (`swingall.jar`) for Java 1.1, included in Java 1.2
- extensive support for sophisticated GUIs (tables, trees, text documents, pluggable look–and–feel)
- BUT: much slower than regular AWT

Why Java?

- Object−Oriented, many packages available
- Interpreted: "byte−code", no linking
- Portable: mostly architecture−independent
- Distributed: sockets, RMI, CORBA
- Simple: no header files, no pointers
- Robust: strongly typed, garbage collection
- Secure: "sandbox" applets, signatures
- Threaded: lightweight processes, easy to code

"Hello World" – Program

```
/* Simple "Hello!" application */

public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello, World");
  }
}
```

Java vs. C – Major Differences

- No headers. No pointers. No macros. No `structs`. No `union`. No `#define`. No `#include`. No `#ifdef`
- No global variables (but public class variables)
- Packages and Imports instead of Includes
- Objects are always passed by reference, no dereferencing!!!
- Garbage collection --> no `malloc()`/`free()`
- Exception handling

Passing by Reference: What does this code do?

```
public void swap (Object a, Object b) {
   Object temp = a;
   a = b;
   b = temp;
}
```

```
import java.util.*;
class Refs {
    public static void main (String[] s) {
        Vector v = new Vector();
        v.addElement(new Integer(3));
        v.addElement(new Integer(4));
        Vector v2 = v;
        System.out.println(v2);
        v.setElementAt(new Integer(100),0);
        System.out.println(v2);
    }
}
```

Data Types

- Primitive Types

Type	Size (bits)
`boolean`	`1`
`char`	`16`
`byte`	`8`
`short`	`16`
`int`	`32`
`long`	`64`
`float`	`32`
`double`	`64`
`array`	`–`

- Object Types

Type	Related
`Boolean`	
`Character`	`String`
`Byte`	
`Short`	
`Integer`	
`Long`	
`Float`	
`Double`	
`–`	`Vector`

Exceptions

- `throws`–Clause. Throwing an exception is signalling an exceptional condition.
- `try/catch`–Statements. Catching an exception is to handle it.
- Pre–Defined Exceptions and User–Defined Exceptions (classes).

Applets vs. Applications

- Applets extend `applet`–Class
  - can be displayed in browsers
  - sandbox principle ––> security issues
- Applications are standalone (command line)
  - started with `java filename` on command line
  - no security restrictions
  - may write GUI using the `java.awt.(J)Frame` and `java.awt.Window` classes

Java Swing (JFC)

- most classes start with a `J` (`JFrame, JApplet, JButton, JPane`…)
- new event model: listeners, callback functions
- separate library (`swingall.jar`) for Java 1.1, included in Java 1.2
- extensive support for sophisticated GUIs (tables, trees, text documents, pluggable look–and–feel)
- BUT: much slower than regular AWT