POWERING POTENTIAL

Introduction to the Java Language

Contents

Objectives

To introduce Java technologies and language

Take a look around Eclipse

Contents

What is Java?

- **Originally named ‘Oak’ (1991) by Sun Microsystems**
- **For enabling devices with different CPUs to share s/w**
- **Was used for web pages with multi-media components**

project lead by ***James Gosling***

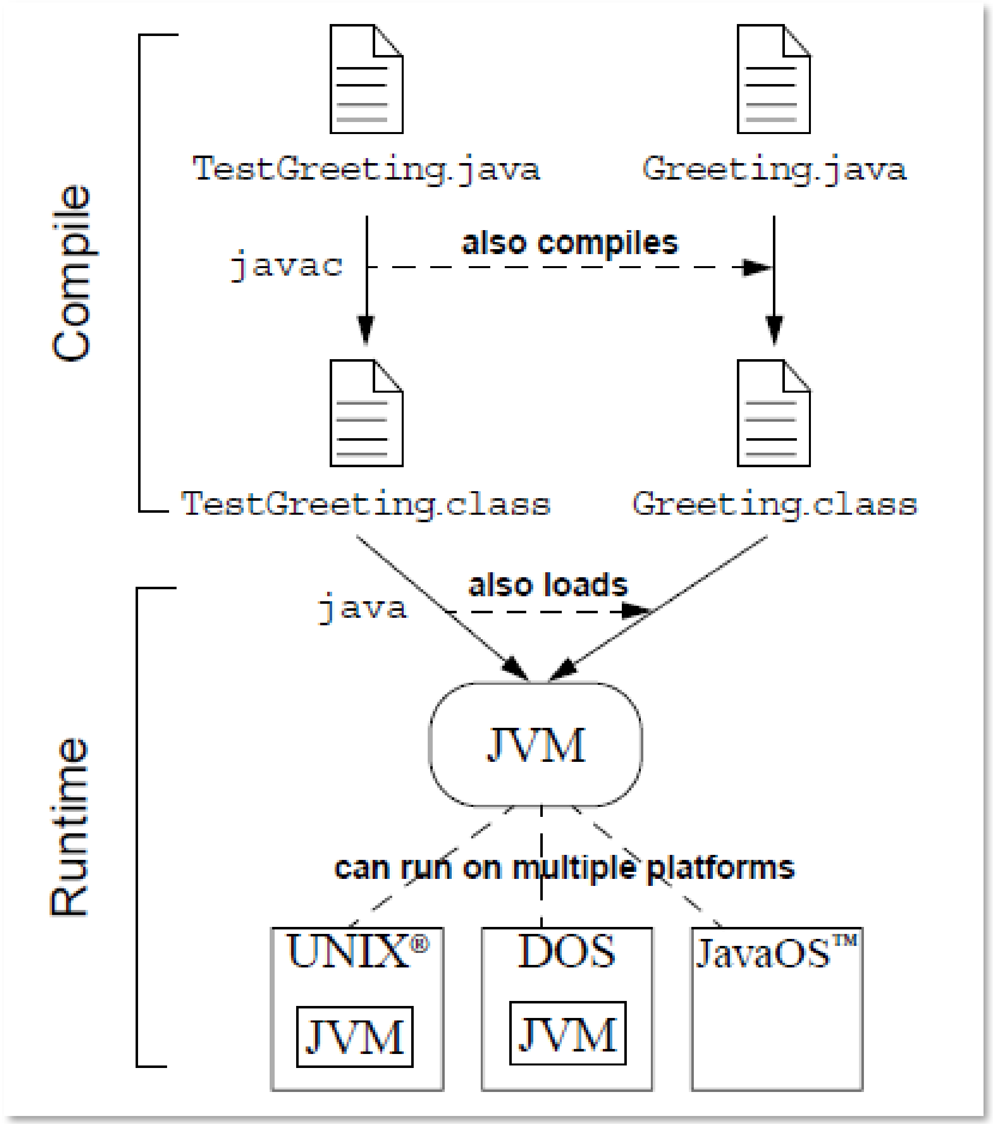
- **So, what is Java?**
  - A programming language
  - A development environment
  - An application environment
  - A deployment environment

Java Development Kit (JDK)

Java runtime Environment (JRE)

Java SDK (Software Development Kit)
Java’s **F**ramework **C**lass **L**ibrary for all apps

Compiling and running a Java app



```
public class Program {
  public static void main(String[ ] args) {
      ...
      ...
  } // end of main method
}
```

Program.class
Byte code

Java packages

Used to group types and avoid naming conflicts

→ Affects location of the compiled code

```
package qa.apprentice;

public class Program {
  public static void main(String[] args ) {
      . . .
  } // end of method Main
}   // end of class Program
```

One of these at top of every file

```
package qa.hr;

public class Timetable {
 // code
}
```

```
package qa.apprentice;

public class Timetable {
 // code
}
```

Java packages and import statements

```
package qa.apprentice;

import java.io.*;

public class Program {
  public static void main(String[ ] args)
  {
    TimeTable qaTimetable;

    FileReader fr;

    ... using ‘fr’ and qaaTimeTable ...
  }
}
```

Java language basics

Variables (symbolic name for an address in memory)

→ Must be declared with a type before use

→ Variables in a method must be initialised before use

```
public void main(String[] args) {
  int myAge;
  boolean answer = true;
  String myName = "Samantha";
  int i = 0, j;

  myAge = 21;
  System.out.println(i); ✓
  System.out.println(j); ✗    // not initialised
}
```

Pre-defined in-built primitive data types

```
byte        eightBit;
short       sixteenBit;
int         thirtyTwoBit;
long        sixtyFourBit;
```

```
float       x32;
double      x64;
```

`Float limits`	**7** digits of precision
`Double limits`	**16** digits of precision

```
Char     initial;    // Unicode character (16 bits)
boolean  isActive;   // can be set to true/false

initial = 'M';
isActive = true;
```

`Java has a special type for large doubled called BigDecimal`
`Please see` https://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html

Standard mathematical operators

```
int x = 4;
x = x + 5;                    // x is 9

double d = 9.0;
d = d / 2;                    // d is 4.5


int y = ( x % 3 );            // y is  1
```

```
int x = 9;
x = x / 2;                    // x is 4!

double d = 9.0;
d = d / 2;                    // d is 4.5
```

+	addition
-	subtraction
*	multiplication
/	division
%	modulus division

Compound operators

- **Each mathematical operator can be combined with '='**

```
int x = 4, y = 6;
x = x + y;              // x = 10
```

```
// Can be coded as
x += y;
```

```
int x = 8;
x *= 2;                // x = 16
```

```
int x = 8;
System.out.println(x % 5);  // displays 3 but x = 8

x %= 5;                     // x = 3
```

Pre- and post-fix ++ and -- operators

```
int x = 0;
x++;                // x = 1
++x;                // x = 2
```

```
int x = 0;
int y = ++x;    // x=1, y=1
```

```
int x = 0;
int y = x++;    // x=1, y=0
```

```
int x = 1;
System.out.println(x++);        // displays 1 and then x=2
System.out.println(x);          // displays 2
```

Casting: Implicit casting

```
int   x = 4;

long no = x;          ✓

double d = x;
```

```
char c = 'A';                        ✓

int   x = c;      65
```

Casting: You can explicitly cast any numeric type to another type

```
double d = 4.5;
int   x = d;
```
✗

```
double d = 4.5;
int   x = (int) d;
```

```
long lng = 5;
int   x = lng;
```
✗

```
long lng = 5;
int   x = (int)lng;
```

```
float f = 4.5;
```
✗

```
float f = (float) 4.5;
      f = 6.75F;
```

```
boolean b = 1;
```
✗

```
boolean b = true;
```

Casting strings

- **Must use a parse method to cast strings to numeric types**

✗
```
String no = "123";
int x = (int)no;
```

```
String no = "123.45";
double d = (double)no;
```

```
int x    = Integer.parseInt(no);

double d = Double.parseDouble(no);

float f  = Float.parseFloat(no);
```
✓

CONDITIONALS

Introducing ‘if’

```
int age = getAge();

if ( age > 18 )
{
 // code for when over 18 years old
}
```

```
int age = getAge();

if ( age < 18) {
  // code for when under 18 years old

}
else {
 // code for when 18 or over

}
```

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

Introducing ‘else if(s)’

```
int mark; . . .

if ( mark > 80 ){
   System.out.println("Distinction");
}
else if( mark > 70 ) {
   System.out.println("Merit");
}
else if( mark > 60 ) {
   System.out.println("Pass");
}
else {
   System.out.println("Try again!");
}
```

`'if'` must come first

as many ‘`else if`'(s) as needed

`'else`’
(if needed) comes last

The ternary conditional operator (? :)

- Produces less code for simple if statements resulting in a value
- These two examples produce the same result

```
double salary = getSalary();

double rate = (salary < 21000) ? 0.2 : 0.4;
```

```
double salary = getSalary();

if(salary < 21000) {
    rate = 0.2;
}
else {
    rate = 0.4;
}
```

Logical operators AND and OR

A

```java
int var1 = 4, var2 = 2, var3 = 0;
if ((var1 > var2) && (var3 == 0))
{
      System.out.println( "Will we see this?" );
}
```

B

```java
int var1 = 4, var2 = 6, var3 = 0;
if ((var1 > var2) || (var3 == 0))
{
      System.out.println( "Will we see this?" );
}
```

C

```java
int var1 = 1, var2 = 2, var3 = 3;
if ((var1 == 1) || (var2 == 2) && (var3 == 1))
{
      System.out.println( "Will we see this?" );
}
```

&&	AND
||	OR
!	NOT

The switch statement

- **Tests an integer, enum, char or String**
- **Statements may be in any order**
- **Often elegant alternative to if…else if… else**
- **Beware of accidentally dropping through to next section**
  → compiler does not force a ‘break’ statement

```
int no = 1, res;

switch ( no ) {

    case 0:
     res = 23;
     break;

    case 2:
     res = 8;
     break;

    case 1: case 2:
     res = 51;
     break;

    default:
     res = -1;
     break;
}
```

METHODS

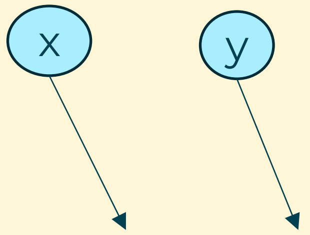
Example of a method with no parameter and no returned value

```
public class Program {
    public static void main(String[] args) {
        updateAllSalaries();
    }

    private static void updateAllSalaries() {
        // Code to update all salaries
    }
}
```

The caller calls the method and lets it do its work!

Example of a void method with two parameters

```java
public class Program {
    public static void main(String[] args) {
        int x = 1, y = 2;
        add(x, y);

        add(3, 7);
    }

    private static void add(int a, int b) {
         System.out.println(a + b);
    }
}
```



Example of method returning a value

```java
public class Program {
    public static void main(String[] args) {
        int x = 1, y = 2;

        int result = getSum(x, y);

        System.out.println("Sum is = ", result);
    }

    private static int getSum(int a, int b) {
        return a + b;
    }
}
```

A method can only return one thing
The caller decides what to do with the returned value

Method overloading

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

        double result1 = getTax(2000);
        System.out.println(result1);

        double result2 = getTax(2000, 0.4);
        System.out.println(result2);
    }

    private static double getTax(double salary) {
         return salary * 0.25;
    }

    private static double getTax(double salary, double rate) {
        return salary * rate;
    }
}
```

Prints **500**

Prints **800**

Same name different parameters

INTRODUCING A FEW JAVA LIBRARY METHODS

Introducing some Java library methods

- Use the `java.util.Scanner` class to input a value

```
Scanner s = new Scanner(System.in);

System.out.println("What is your name?");
String name = s.nextLine();

System.out.println("What is your age?");
int age = s.nextInt();

System.out.println( "Hi " + name + "\n next year you'll be " + (age + 1));
```

Printing formatted output

`System.out.println(..)` **is hugely overloaded**

→ Can build a ‘String’ via concatenation (using +).
This is onerous and error prone (spacing)

```
int age = 21;
String name = "Bob";
System.out.println("Hi " + name + ", next year you will be " + age + 1);
```

Hi Bob, next year you will be 211

→ Best use **`printf`**() and **String.`format`**() `for formatting`

```
System.out.printf("Hi %s, next year you will be %d”, name, age + 1);
```

Format string

Examples: printf()

- **%s** used to represent a String

```
String word = "wibble";
System.out.printf("My favourite word is %s\n", word);
```

%d	for an int value
%8d	len 8, right justified
%08d	with leading zeros
%+8d	include sign +/-
%,8d	thousand separator

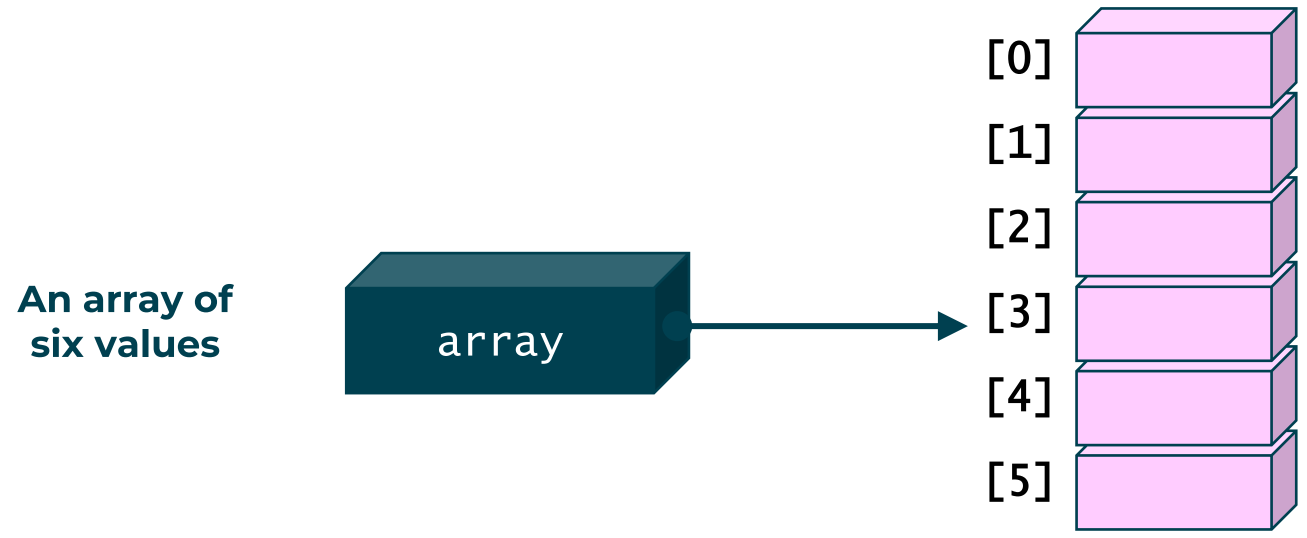
```
int num = 123456;
System.out.printf("%d\n",     num);   // --> “123456"

System.out.printf("%08d\n",   num);   // --> "00123456"

System.out.printf("%+8d\n",   num);   // --> " +123456"

System.out.printf("%,8d\n",   num);   // --> " 123,456"

System.out.printf("%+,8d\n", num);   // --> "+123,456"
```

ARRAYS AND LOOPS

WHAT IS AN ARRAY?

- **An array can store a collection of variables all of the same type**
  → Each array element can hold a single item (value / reference type)
  → Array elements are accessed by index number, e.g., names[3]

- **Arrays are objects**
  → Must be created before they can be used
  → An array variable (of any type) is a reference type

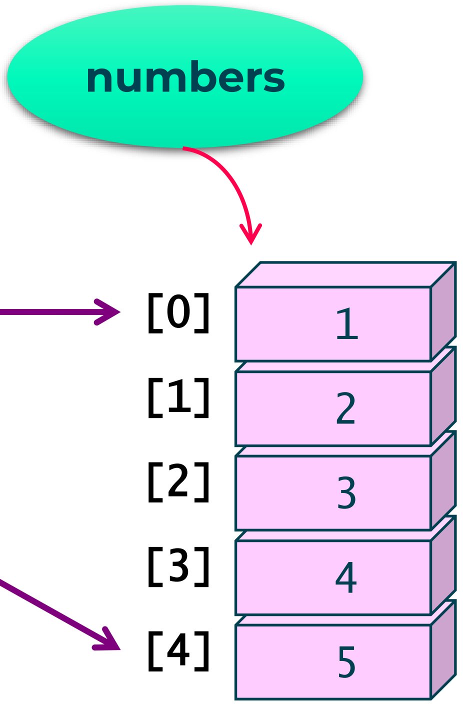


Introduction to arrays

- **Array is a fixed-size collection of elements of the same data type**

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

System.out.println(numbers[0]);

System.out.println(numbers[4]);

System.out.println(numbers.length);
```

5

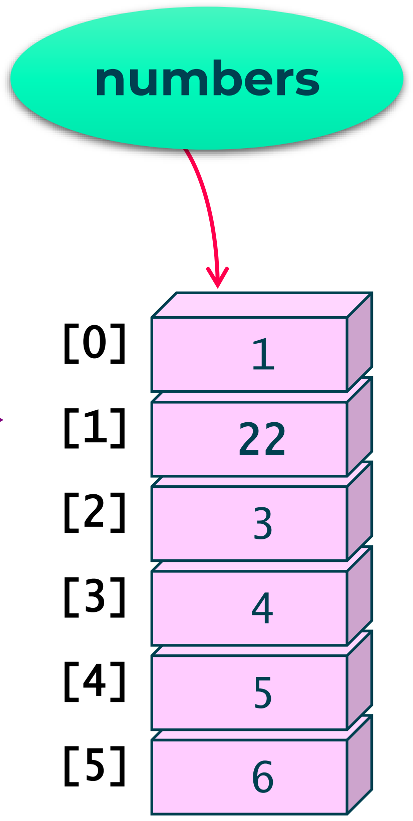


Introduction to arrays

- You can change array elements
- Appending, inserting, and removing an element is hard

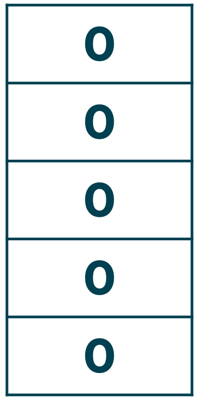
```
int[] numbers = {1,2,3,4,5};

numbers[1] = 22;

System.out.println(numbers[2]);
```

```
22
```



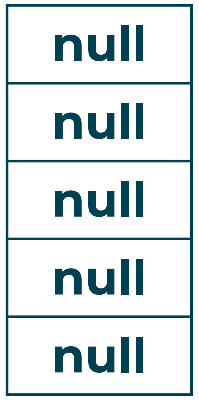
Create an empty array

```
int[] numbers = new int[5];
```



Initialised to default values for int

```
String[] names = new String[5];
```



Initialised to default values for String

LOOPS

- **Objectives**
  - To cover Java’s looping constructs

Iteration using while and for statements

```
while ( boolean_expression ) {
  statement(s);
}
```

```
do {
  statement(s);
} while ( boolean_expression );
```

```
for ( init_expr; boolean_expr; update_expr ) {
  statement(s);
}
```

We'll see iteration using enhanced for loops later

Iteration using while loops

Declare / Init

Exit condition

```
int i = 0;
while (i < 10)
{
   System.out.printf("%d squared = %d\n", i, i*i );
   i++;
}
```

Changing value

→ or put the test condition at the end of the loop

```
int i = 0;
do
{
   System.out.printf("%d squared = %d\n", i, i*i );
   i++;
} while (i < 10);
```

The for loop

Initialisation can include a declaration

→ Declared variable is in scope only inside the loop

Initialisation Exit condition Changing

```
for ( int i = 0; i < 10; i++ ) {
   System.out.println(i + "\n" );
}
```

what is the value of 'i' here?

Initialisation and update can be a list of ',' separated expressions

```
for( int i = 0, j = 10; i < j; i++, j-- ) {
   System.out.println( i * j + "\n");
}
```

Using break to exit a any loop

```
double money = 50;          // £ pounds
double interest = 0.06;     // percent

for (int years = 1; money < 1000; years++) {
     money += (money * interest);
     print("Year : "+ years + ": " + money);

     double tax = money * 0.40;
     if (tax > 100) {
          print("Tax is > 100");
          break;
     }
}
```

Break out of the current loop

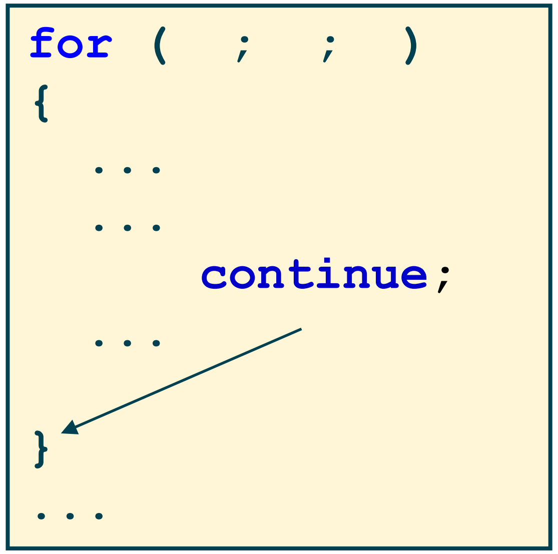
```
Year 1: 53
Year 2: 56
Year 3: 59
Year 4: 63
Year 5: 66
Year 6: 70
Year 7: 75
Year 8: 79
Year 9: 84
Year 10: 89
Year 11: 94
Year 12: 100
Year 13: 106
Year 14: 113
Year 15: 119
Year 16: 127
Year 17: 134
Year 18: 142
Year 19: 151
Year 20: 160
Year 21: 169
Year 22: 180
Year 23: 190
Year 24: 202
Year 25: 214
Year 26: 227
Year 27: 241
Year 28: 255
Tax is > 100
```

Java

```
outer_loop:
for (  ;  ;  ) {
 for (  ;  ;  ) {
  if (...) {
    break outer_loop;
  }
 }
}
...
```

Break out of nested loops

Continue



```
for (   ;   ;   )
{
   ...
   ...
         continue;
   ...

}
...
```

```
for ( int i = 0; i < 10; i++ ) {
  if ( i % 4 == 0) {
     // few statements
     continue;
  }
  // many statements
}
```

```
for ( int i = 0; i < 10; i++ ) {
  if ( i % 4 == 0) {
    // few statements
  }
  else {
    // many statements
  }
  // no code here!!
}
```

Can be coded as

The enhanced for loop

For iterating over a collection or an array without testing for the bounds

```
public void processNames( String[] names ) {
  for (String name : names ) {
    System.out.println( name );
  }
}
```

Read as: foreach string 'name' in the 'names' collection

```
String[] names = {"Bob","Sasha"};
for (String name : names) {
        name += "x";
}
System.out.println(names[0]);
```

Bob

The elements are considered read-only

Which iteration statement?

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

int i = 0;

While(i < numbers.length) {
      System.out.println(numbers[i]);
      i++;
}
```

```
for (int i = 0; i < numbers.length; i++) {
        System.out.println(numbers[i]);
}
```

```
for(int no : numbers) {
   System.out.println(no);
}
```

Review

In this chapter we reviewed:

Creating variables

Selection statements
if, else, else if, and switch

Iteration statements
while, do while, for, and enhanced for loops

Creating and using methods

Images by Storyset

Lab

Practise the basics of the Java language

Duration 1.5 hour

Operator precedence

Order	Operators	Comments
1	`( ) . f(x) [ ] x++ x– new`	Primary
2	`+ - ! ~ ++x --x (T)x`	Unary
3	`* / %`	Multiplicative
4	`+ -`	Additive
5	`<< >>`	Bit Shift
6	`< > <= >= instanceof`	Relational
7	`== !=`	Equality
8	`&`	Bitwise AND
9	`^`	Bitwise exclusive OR
10	`|`	Bitwise inclusive OR
11	`&&`	Logical AND
12	`||`	Logical OR
13	`?:`	Conditional
14	`= op=`	Assignment

```
Vehicle v = ...;
((Car)v).openSunRoof();
```

Much more later ..

```
int p = x + (y * z);
int q = (x + y) * z;
bool b1, b2, b3;
if (b1 && b2 || b3) {
  //does this if b3 true
  //b1 and/or b2 might be
}
```

```
while((b = getNextByte()) != -1) {...}
```