Introduction to Java and Agent-Based Economic Platforms (CF-904)

1. Introduction to Java for JAS

Mr. Simone Giansante
Email: sgians@essex.ac.uk
Web: http://privatewww.essex.ac.uk/~sgians/
Office: 3A.531

University of Essex
CCFEA

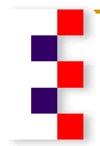
Syllabus

- **Aims**: the student will be taught how to build, debug and run JAS models and also learn how to access JAS libraries that simplify the programming needed to set up multi-agent economic models and to display the results.

- **Method**: a step by step method will be used for the student to learn how to prepare and implement a number of specific ACE models.

Lab timetable

- **Consisting in 2-hour laboratories**
  - Weeks 3-11
  - Thursday 3pm – 5pm
- **All the material will be provided on the CF-902 web page**
- **Recommended readings:**
  - Handbook of Computational Economics, Vol 2: Agent-Based Computational Economics, published by Elsevier/North-Holland (Handbooks in Economics Series), May 2006
  - Sun Microsystems, The Java Tutorial, http://java.sun.com/docs/books/tutorial/index.html



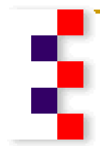
Software

- Java Editor: Eclipse
  - Web: www.eclipse.com
  - CCFEA server:

    \\sernt2\ccfea-lab$\eclipse 3.2\eclipse.exe

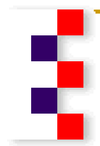
- Agent-Based Simulator: JAS
  - Web: http://jaslibrary.sourceforge.net
  - CCFEA server:

    \\sernt2\ccfea-lab$\JAS 1.2\JAS-1.2.1.jar



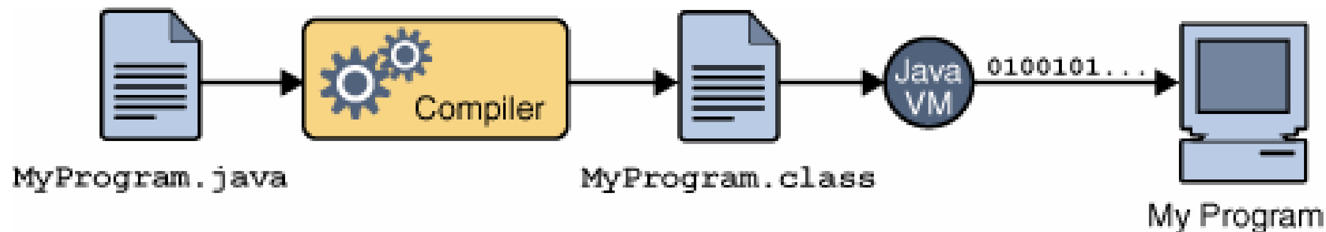
Java features

- Simple
- Object oriented
- Distributed
- Multithreaded
- Dynamic
- Architecture neutral
- Portable
- High performance
- Robust
- Secure



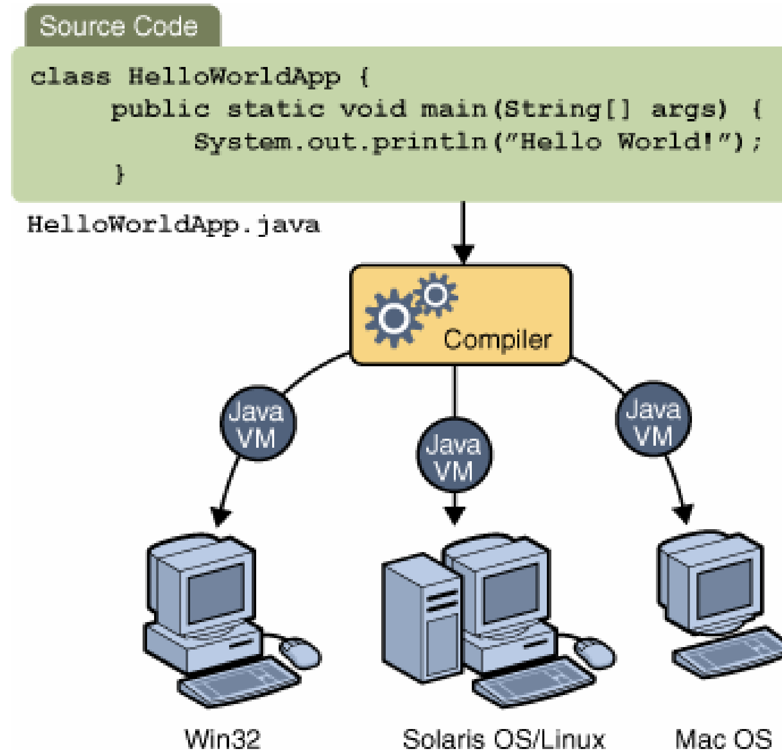
The Java Programming Language

- The source code is written in text files with extension .java
- They are compiled in .class files by the javac compiler
- The binary codes in the .class files are converted in native code of your processor by the Java Virtual Machine (Java VM)



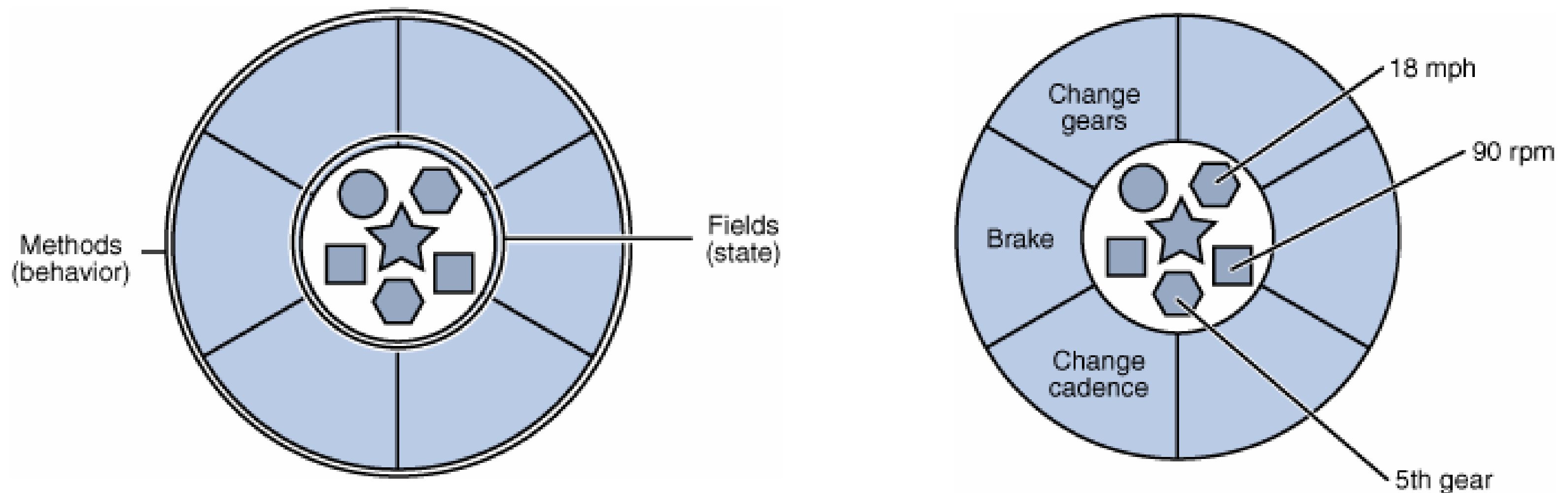
Java VM

As Java VM is available on many different operating systems, the same .class files are capable of running on multiple platforms



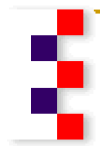
Object-Oriented Programming

- Objects have two characteristics:
  - State (Fields or Variables)
  - Behavior (Methods)



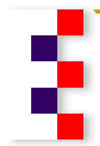
Variables of a primitive type (1)

byte	8-bit	-128	127
short	16-bit	-32,768	32767
int	32-bit	-2,147,483,648	2,147,483,647
long	64-bit	-9,223e^9	9,223e^9
float	32-bit	4 bytes of storage, 23 binary digits of precision	
double	64-bit	7 bytes of storage, 52 binary digits of precision	
boolean	1-bit	false	true
char	16-bit	--	



Variables of a primitive type (2)

- Declaration:
  - boolean result = true;
  - char capitalC = 'C';
  - byte b = 100;
  - short s = 10000;
  - int i = 100000;
  - double d1 = 123.4;
  - double d2 = 1.234e2; // same value as d1, but in //scientific notation
  - float f1 = 123.4f;



Generic variables

- Every variable, a part of those that refer to primitive types, is an instance of a specific class. The declaration is the following:

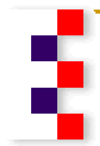
  type NameOfTheVariable;

  i.e.: Agent agent;

- The creation of variables, in order to allocate memory, requires the operator *new*

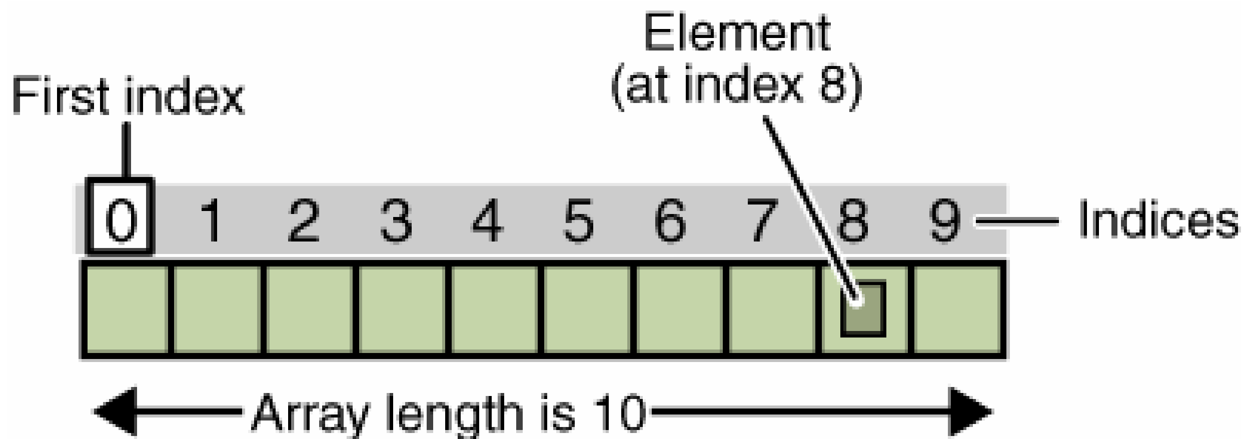
  NameOfTheVariable = ***new*** *type ();*

  i.e.: agent = ***new*** *Agent ();*



Arrays (1)

- An *array* is a container object that holds a fixed number of values of a single type
- each item in an array is called an *element*



Arrays (2)

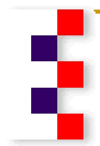
```
int [ ] anArray;  // declare an array of integers

anArray = new int[10]; // create an array of integers

anArray[0] = 100; // initialize first element
anArray[1] = 200; // initialize second element
anArray[2] = 300; // etc.
```

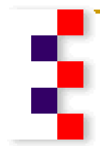
shortcut syntax:

```
int [ ] anArray = {100, 200, 300, 400, 500};
```



Operators (1)

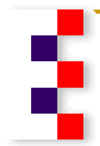
- **The Arithmetic Operators**:
  - **+** additive operator (also used for String concatenation)
  - **-** subtraction operator
  - ***** multiplication operator
  - **/** division operator
  - **%** remainder operator



Operators (2)

- **Example 1:**

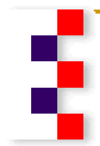
```
public static void main (String[] args){
        int result = 1 + 2; // result is now 3
        System.out.println(result);
        result = result - 1; // result is now 2
        System.out.println(result);
        result = result * 2; // result is now 4
        System.out.println(result);
        result = result / 2; // result is now 2
        System.out.println(result);
        result = result + 8; // result is now 10
        result = result % 7; // result is now 3
        System.out.println(result);
}
```



Operators (3)

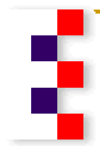
- **Example 2:**

```
public static void main(String[] args){
        String firstString = "This is";
        String secondString = " a concatenated string.";
        String thirdString = firstString+secondString;
        System.out.println(thirdString);
}
```



Operators (4)

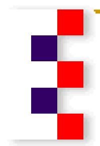
- **The Unary Operators**:
  - **+** Unary plus operator; indicates positive value
  - **-** Unary minus operator; negates an expression
  - **++** Increment operator; increments a value by 1
  - **--** Decrement operator; decrements a value by 1
  - **!** Logical complement operator; inverts the value of a boolean



Operators (4)

- Example 1:

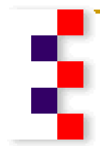
```
public static void main(String[] args){
        int result = +1; // result is now 1
        System.out.println(result);
        result--; // result is now 0
        System.out.println(result);
        result++; // result is now 1
        System.out.println(result);
        result = -result; // result is now -1
        System.out.println(result);
        boolean success = false;
        System.out.println(success); // false
        System.out.println(!success); // true
}
```



Operators (5)

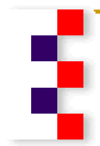
- Example 2:

```
public static void main(String[] args){
        int i = 3;
        i++;
        System.out.println(i); // "4"
        ++i;
        System.out.println(i); // "5"
        System.out.println(++i); // "6"
        System.out.println(i++); // "6"
        System.out.println(i); // "7"
}
```



Operators (6)

- **Equality and Relational Operators:**
  - **==** equal to
  - **!=** not equal to
  - **>** greater than
  - **>=** greater than or equal to
  - **<** less than
  - **<=** less than or equal to



Operators (7)

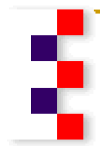
- Example:

```
public static void main(String[] args){
	int value1 = 1;
	int value2 = 2;
	if(value1 == value2) System.out.println("value1 == value2");
	if(value1 != value2) System.out.println("value1 != value2");
	if(value1 > value2) System.out.println("value1 > value2");
	if(value1 < value2) System.out.println("value1 < value2");
	if(value1 <= value2) System.out.println("value1 <= value2");
}
```

Operators (8)

- **Conditional Operators**
  - **&&** Conditional-AND
  - **||** Conditional-OR
- Example:

```
public static void main(String[] args){
  int value1 = 1; int value2 = 2;
  if((value1 == 1) && (value2 == 2)) System.out.println("value1 is 1 AND value2 is 2");
  if((value1 == 1) || (value2 == 1)) System.out.println("value1 is 1 OR value2 is 1");
}
```

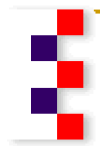


Control Flow Statements (1)

- **if-then statement**

The if-then statement tells your program to execute a certain section of code *only if* a particular test evaluates to true.

```
void applyBrakes()
{
        if (isMoving)                    // the "if" clause: bicycle must moving
        {
                currentSpeed--;          // the "then" clause: decrease current speed
        }
}
```

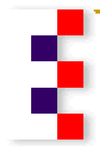


Control Flow Statements (2)

- **if-then-else statement**

The if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false.

```
void applyBrakes(){
	if (isMoving) {
		currentSpeed--;
	}
	else {
		System.err.println("The bicycle has already stopped!");
	}
}
```

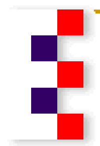


Control Flow Statements (3)

- **switch statement**

  Unlike if-then and if-then-else, the switch statement allows for any number of possible execution paths.

```
public static void main(String[] args) {
        int month = 8; switch (month) {
                case 1:   System.out.println("January"); break;
                case 2:   System.out.println("February"); break;
                case 3:   System.out.println("March"); break;
                case 4:   System.out.println("April"); break;
                case 5:   System.out.println("May"); break;
                case 6:   System.out.println("June"); break;
                case 7:   System.out.println("July"); break;
                case 8:   System.out.println("August"); break;
                case 9:   System.out.println("September"); break;
                case 10: System.out.println("October"); break;
                case 11: System.out.println("November"); break;
                case 12: System.out.println("December"); break;
                default:   System.out.println("Invalid month.");break;
        }
}
```

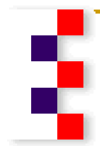


Control Flow Statements (4)

- **while statement**

The while statement continually executes a block of statements while a particular condition is true.

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

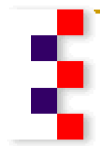


Control Flow Statements (5)

- **do-while statement**

The do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once.

```
public static void main(String[ ] args){
        int count = 1;
        do {
                System.out.println("Count is: " + count); count++;
        } while (count <= 11);
}
```



Control Flow Statements (6)

- **for statement**

  The for statement provides a compact way to iterate over a range of values.

```
for (initialization; termination; increment)
{
        statement(s);
}
```

- **example:**

```
public static void main(String[] args){
    for(int i=1; i<11; i++){
        System.out.println("Count is: " + i);
    }
}
```