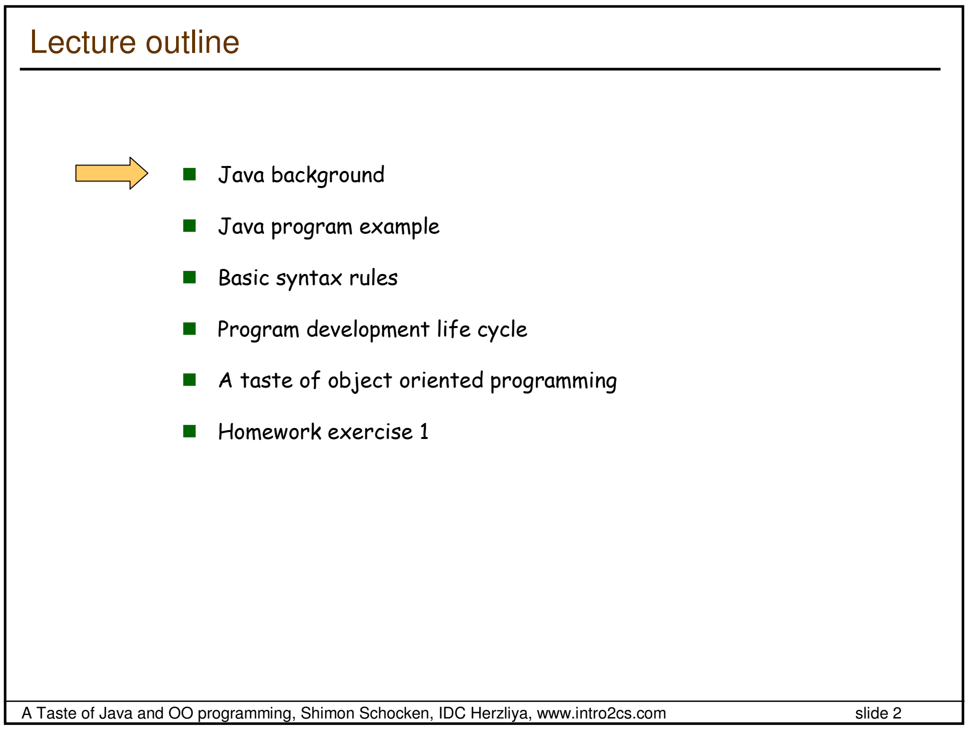
Introduction to Computer Science
Shimon Schocken
IDC Herzliya

Lecture 1-2:

Lecture 1-2:

A Taste of Java and Object-Oriented Programming



Lecture outline

- Java background
- Java program example
- Basic syntax rules
- Program development life cycle
- A taste of object oriented programming
- Homework exercise 1

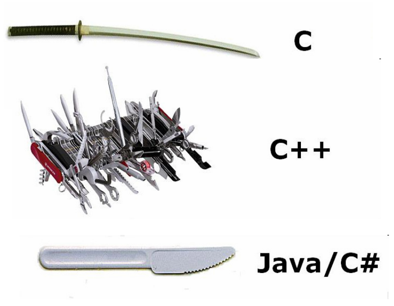
Java background

- Brief history:
- 1995: invented by James Gosling at Sun Microsystems (1995)
  - Original design objective:
    a programming language for the Internet: safety and portability
  - Actual accomplishment: a great language in almost every respect
  - Borrows from many other languages:
    C / C++, Pascal, Scheme, SmallTalk
- 1996: IDC adopts Java as CS101 programming language
- 1999: Microsoft releases C#

Why did we adopt Java?

Because Java ...

- Is object oriented (OO)
- Encourages good programming habits
- Similar to C++, but simpler and more elegant
- Commercial
- Cool.



Java program example

Task: Print the numbers 0 to 5

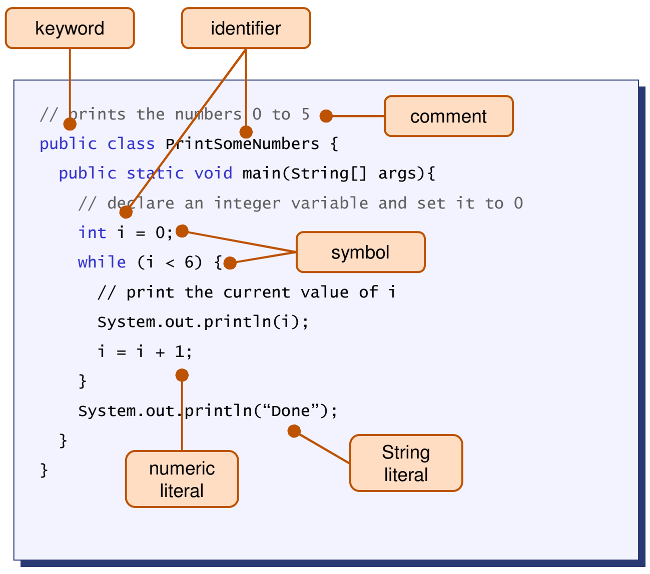
Algorithm:

```
i = 0;
while (i < 6)
  print i
  i = i + 1
```

Java implementation:

```
// prints the numbers 0 to 5
public class PrintSomeNumbers {
  public static void main(String[] args){
    // declare an integer variable and set it to 0
    int i = 0;
    while (i < 6) {
      // print the current value of i
      System.out.println(i);
      i = i + 1;
    }
    System.out.println("Done");
  }
}
```

Java syntax elements (first approximation)



```
// prints the numbers 0 to 5
public class PrintSomeNumbers {
  public static void main(String[] args){
    // declare an integer variable and set it to 0
    int i = 0;
    while (i < 6) {
      // print the current value of i
      System.out.println(i);
      i = i + 1;
    }
    System.out.println("Done");
  }
}
```

"Words":
- Reserved words
- Identifiers

Literals:
- Numbers
- Strings
- (More later)

Symbols:
- ( ) [ ] { } ,
  . ; + - * / …

Comments
- Text beginning with //

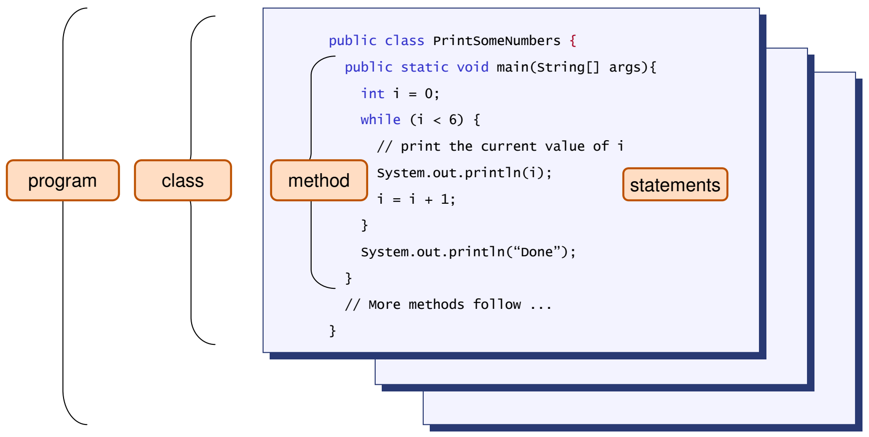
Java reserved words

abstract	continue	for	new	switch
assert	default	goto(*)	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strict	volatile
const(*)	float	native	super	while

(*): Not used.

Java program structure

- Program (loosely defined): consists of one or more classes
- Class: consists of one or more methods, one of which must be named `Main()`
- Method: a sequence of statements
- Statement: ends with a semicolon (;) or enclosed in curly braces ({ })



```
public class PrintSomeNumbers {
  public static void main(String[] args){
    int i = 0;
    while (i < 6) {
      // print the current value of i
      System.out.println(i);
      i = i + 1;
    }
    System.out.println(“Done”);
  }
  // More methods follow ...
}
```

White space

```
// prints the numbers 0 to 5
public class PrintSomeNumbers {
  public static void main(String[] args){
    // declare an integer variable and set it to 0
    int i = 0;
    while (i < 6) {
      // print the current value of i
      System.out.println(i);
      i = i + 1;
    }
    System.out.println(“Done”);
  }
}
```

Same functionality

```
public class PrintSomeNumbers {public static void main(String[] args){int i=0;while
(i<5){System.out.println(i);i=i+1;}System.out.println(“Done”);}}
```

White space = comments and indentation (ignored by the compiler).
White space, left to the programmer’s discretion, is used for readability

Purpose: To make programs readable

Important: Program readability and clarity are as important as program correctness (maybe more)!

Syntax / semantics / style

Syntax: the rules of the language: vocabulary and grammar

Semantics: what a sentence in the language means

Style: how well do you say it?

Natural languages:

Sometimes it is allowed to break the syntax rules

Occasionally there is more than one meaning to a sentence.

Programming languages:

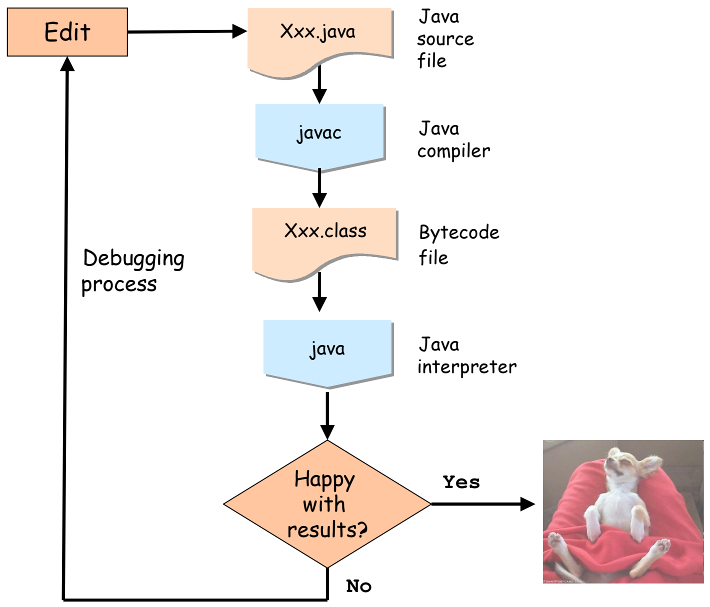
You are never allowed to break the syntax rules

There is only one semantic interpretation: no ambiguity.

Lecture outline

- Java background
- Java program example
- Basic syntax rules
- Program development life cycle
- A taste of object oriented programming
- Homework exercise 1

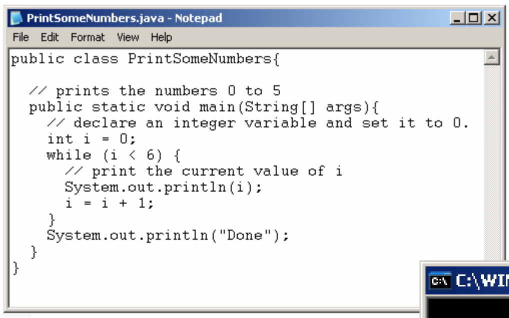
Java program development life cycle



A Taste of Java and OO programming, Shimon Schocken, IDC Herzliya, www.intro2cs.com slide 11

The tools of the trade: basics

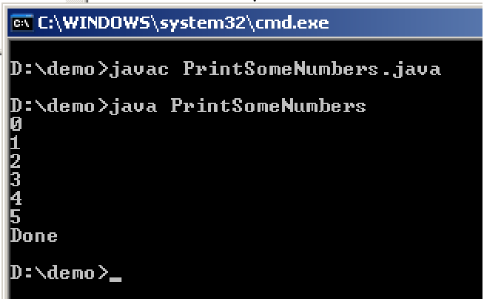
Plain text editor



```
public class PrintSomeNumbers{

  // prints the numbers 0 to 5
  public static void main(String[] args){
    // declare an integer variable and set it to 0.
    int i = 0;
    while (i < 6) {
      // print the current value of i
      System.out.println(i);
      i = i + 1;
    }
    System.out.println("Done");
  }
}
```

Compilation and execution:



```
D:\demo>javac PrintSomeNumbers.java

D:\demo>java PrintSomeNumbers
0
1
2
3
4
5
Done

D:\demo>_
```

Debugging

0. Run the program
1. Observe the program's execution
2. Figure out what's wrong
3. Fix the code in the editor
4. Goto step 0 …

A Taste of Java and OO programming, Shimon Schocken, IDC Herzliya, www.intro2cs.com slide 12

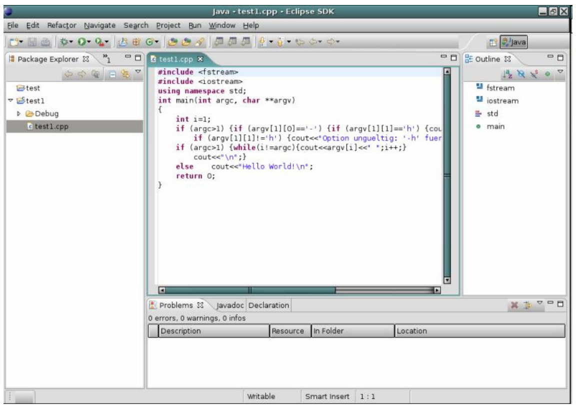
The tools of the trade: Integrated Development Environments

IDE: a software package that features a combination of:

- Editor (programming-oriented)
- Compiler
- Debugger
- Project Manager
- Many more cool goodies

Some Commercial IDEs:

- Eclipse (open source)
- Visual Age
- InteliJ
- Jcreator
- NetBeans
- . . .



```
#include <fstream>
#include <iostream>
using namespace std;
int main(int argc, char **argv)
{
    int i=1;
    if (argc>1) {if (argv[1][0]=='-') {if (argv[1][1]=='h') {cou
        if (argv[1][1]!='h') {cout<<"Option ungueltig: '-h' fuer
    if (argc>1) {while(i!=argc){cout<<argv[i]<<" ";i++;}
        cout<<"\n";}
    else    cout<<"Hello World!\n";
    return 0;
}
```

0 errors, 0 warnings, 0 infos

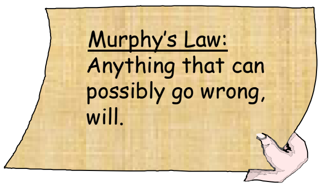
Description	Resource	In Folder	Location

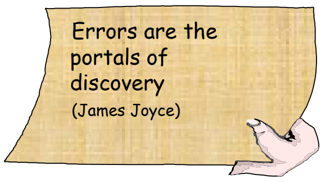
Debugging

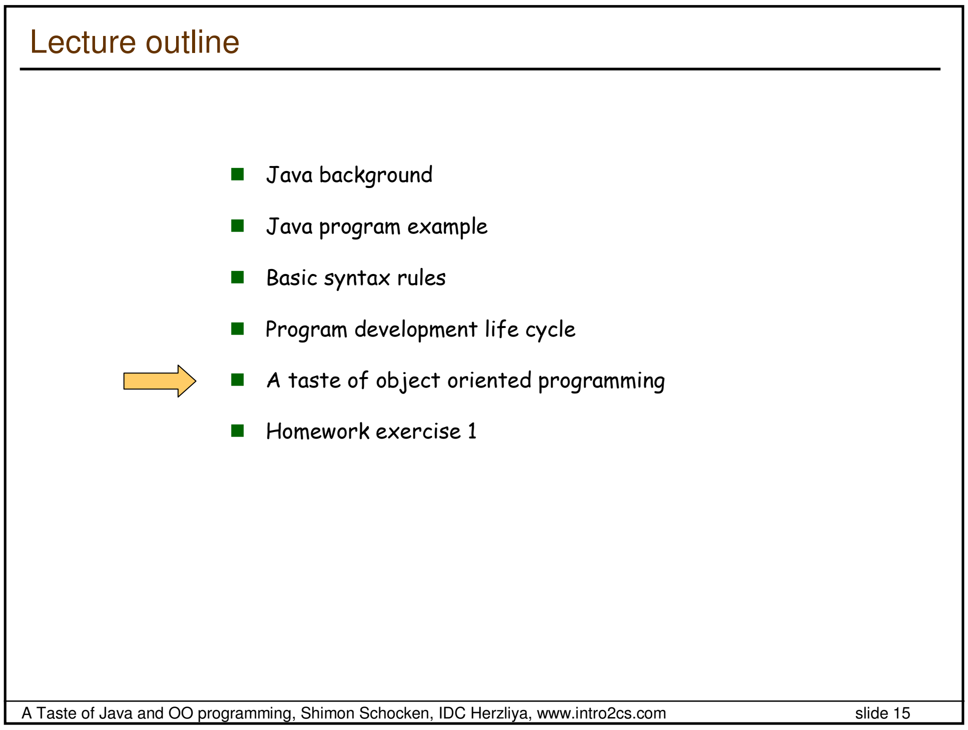
That's what you'll do most of the semester

Error types:

- **Compile-time errors:** mostly syntax violations;
  detected by the compiler
- **Run-time errors:** the program passes compilation,
  runs, but crashes
- **Logical errors:**
  - ❑ The program runs, doing something you didn't want it to do
  - ❑ The program runs, but should be improved for some reason.

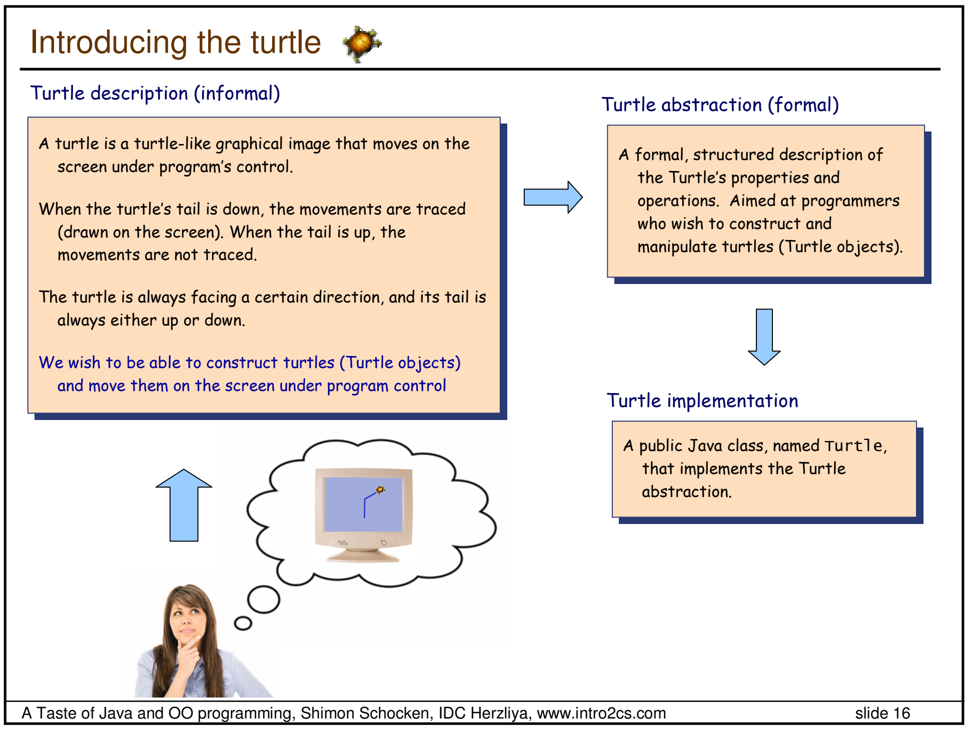






Lecture outline

- Java background
- Java program example
- Basic syntax rules
- Program development life cycle
- A taste of object oriented programming
- Homework exercise 1



Introducing the turtle

Turtle description (informal)

A turtle is a turtle-like graphical image that moves on the screen under program's control.

When the turtle's tail is down, the movements are traced (drawn on the screen). When the tail is up, the movements are not traced.

The turtle is always facing a certain direction, and its tail is always either up or down.

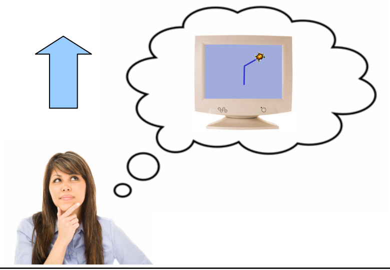
We wish to be able to construct turtles (Turtle objects) and move them on the screen under program control

Turtle abstraction (formal)

A formal, structured description of the Turtle's properties and operations. Aimed at programmers who wish to construct and manipulate turtles (Turtle objects).

Turtle implementation

A public Java class, named `Turtle`, that implements the Turtle abstraction.



Turtle abstraction = Turtle class interface = Turtle API

Turtle class API (partial)

Constructor Summary
`Turtle()` Constructs a new turtle.

Method Summary	
`void`	`disableDelay()` Disables the delay of the turtle.
`void`	`hide()` Hides the turtle.
`void`	`moveBackward(double units)` Moves the turtle backwards by a given number of units.
`void`	`moveForward(double units)` Advances the turtle forwards by a given number of units.
`void`	`show()` Shows the turtle.
`void`	`tailDown()` Lowers the tail of the turtle.
`void`	`tailUp()` Raises the tail of the turtle.
`void`	`turnLeft(int degrees)` Turns the turtle counter-clockwise.
`void`	`turnRight(int degrees)` Turns the turtle clockwise.

- The `Turtle` implementation is a black box: we have no access to its code
- The `Turtle` abstraction (API) is a publicly available document
- The API specifies which operations can be invoked on `Turtle` objects, and how to invoke them
- Some of these operations are designed to create new `Turtle` objects, while others are designed to manipulate existing Turtle objects

OOP Terminology: The words

- *Abstraction*
- *Class interface*
- *API*

Mean the same thing: a structured, agreed-upon, user-oriented way to document class functionality.

Using the Turtle API

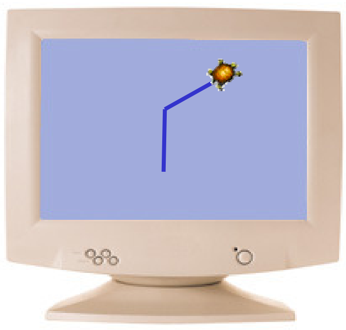
Turtle class API (partial)

Constructor Summary
`Turtle()` Constructs a new turtle.

Method Summary	
`void`	`disableDelay()` Disables the delay of the turtle.
`void`	`hide()` Hides the turtle.
`void`	`moveBackward(double units)` Moves the turtle backwards by a given number of units.
`void`	`moveForward(double units)` Advances the turtle forwards by a given number of units.
`void`	`show()` Shows the turtle.
`void`	`tailDown()` Lowers the tail of the turtle.
`void`	`tailUp()` Raises the tail of the turtle.
`void`	`turnLeft(int degrees)` Turns the turtle counter-clockwise.
`void`	`turnRight(int degrees)` Turns the turtle clockwise.

Turtle usage example

```
public class TurtleDrawingDemo {
  public static void main(String[] args){
    Turtle leonardo = new Turtle();
    leonardo.tailDown();
    leonardo.moveForward(100);
    leonardo.turnRight(60);
    leonardo.moveForward(100);
  }
}
```



Object oriented programming

In OOP, much of the programming activity evolves around creating and manipulating objects of certain *types*. For example, `leonardo` is an object of type `Turtle`

The rules for creating and manipulating objects are specified in class interfaces

Some of these classes are implemented by you; some classes come from the Java class library; some are implemented by other programmers who you may or may not know

For example, if someone wrote a class named `BouncingBall` and made it publicly available, programmers who develop applications that need bouncing ball functionality can now use the `BouncingBall` API

<u>Some OOP advantages</u>

- Code reuse: no need to re-invent the wheel
- Code consistency
- Divide and conquer
- Modularity.

Homework Exercise 1

- Play with a simple Java program
- Experience debugging
- Do some turtle graphics
- Further instructions: see the course web site.