Data Structures and Algorithms

Lecture 4

Mark Cudden
Mark.Cudden@ncirl.ie

4. Abstract Data Types: STACK

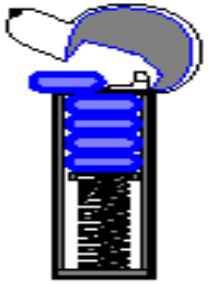
4.1 Abstract Data Types: Introduction

- In Object Oriented (OO) programming, encapsulation is the inclusion within a program object of all the resources needed for the object to function - basically the methods and the data members (attributes).
- The visible part of your object are public methods and data members – but objects should keep their data members private.
- The idea is "don't tell me how you do it; just do it."

4.1 Abstract Data Types: Introduction

- Data abstraction asks that you to think in terms of what you can do to a collection of data independently of how you do it.
- An ADT - Abstract Data Type is
  - a collection of data and
  - a set of operations on that data.
- A data structure is a construct within a programming language that stores a collection of data.
  - ArrayList, Vector
  - Stack
  - Queue

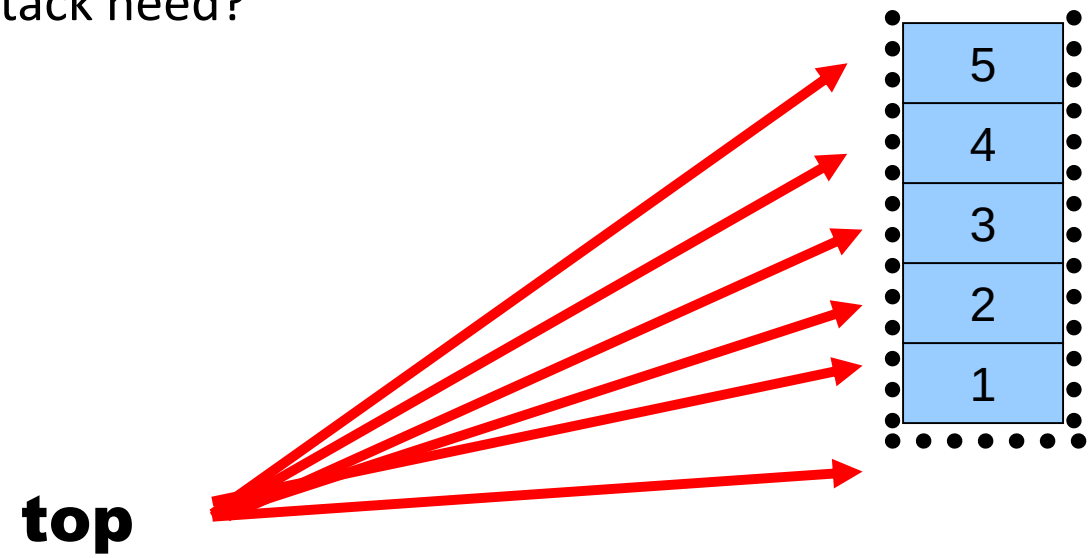
4.2. Stack – LIFO Principle



- A stack is a container of objects that are inserted and removed according to the last-in-first-out (LIFO) principle
- Objects can be inserted at any time, but only the last (the most-recently inserted) object can be removed
  - "**pushing**": inserting an item onto the stack. Objects are added to the top of the stack
  - "**popping**": removing an item off the stack. Only the item from the top of the stack can be accessed

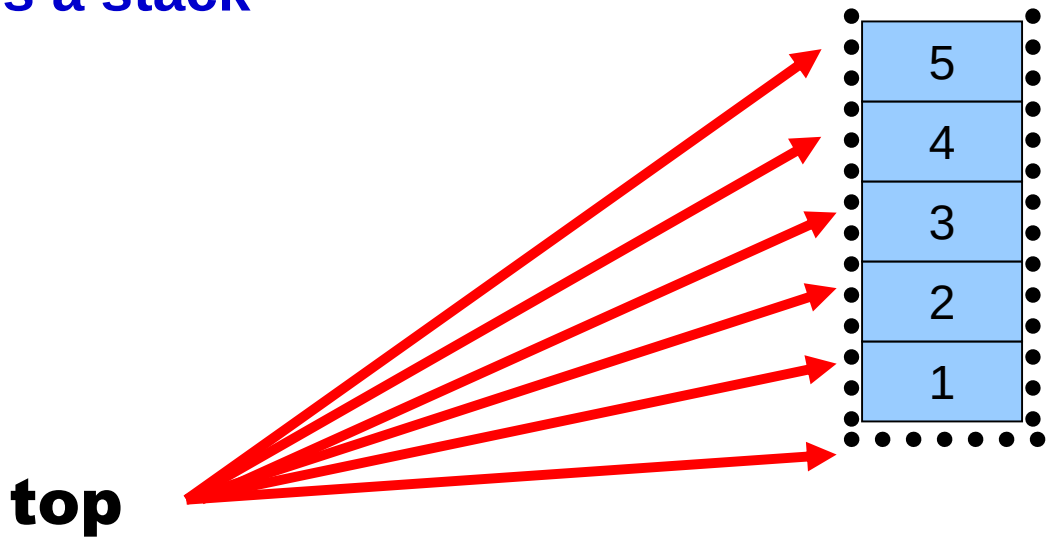
4.3 Stack Animation 1

- Examples
  - a stack of paper
  - A stack of plates
- What methods does a stack need?
  - **Push** - adding



4.3 Stack Animation 2

- **Examples**
  - a stack of paper
  - A stack of plates
- **What methods does a stack need?**
  - **Pop** - removing



4.4 Stack Operations

- A stack is seen as a linear list in which insertions (pushes) and removals (pops) take place at the same end.
- This end is called the top, the other end being the bottom.
- Operations defined with a stack:
  - **push** – insert an element onto the stack
  - **isEmpty** – is the stack empty
  - **pop** – remove an element from the stack

4.5 Stack Exercise

- Outline the state of an initially empty stack, and then after each of the following operations:
  - push(3)
  - pop()
  - pop()
  - push(2)
  - push(1)
  - push(11)
  - pop()
  - pop()

4.6 The Stack ADT

Mathematical Model:

Collection (linear list) of elements that are inserted and removed according to the Last In First Out - LIFO principle.

Operations: (common ones listed)

- **`isEmpty()`** – is the stack empty
- **`size()`** – the number of objects in the stack
- **`push(Object o)`** – insert an element onto the stack
- **`pop()`** – remove an element from the (error if empty)

4.7 Interface

- Interfaces are reference types like classes
- Can have only abstract method declarations and constants as members.
- Stored in .java files
- Java alternative for multiple inheritance – a Java class can implement multiple interfaces.
- An object of this class then has a sort of alternate identity and can be referred to as being of the interface type.
- A class that implements an interface must provide implementations for all of the methods that the interface defines.

```
public class myClass implements Interface
```

4.8 Stack Interface

- The Stack ADT can be defined as a Java interface:

```
public interface StackInterface {
 public boolean isEmpty();
 public boolean isFull();
 public void push(Object newItem);
 public Object pop();
}
```

- We have now defined the ADT, but how do we go about implementing it?

- Object type
  - All Java objects derive from the Object base class.

4.8 Stack Interface

- Object class
  - This class sits at the top of the class hierarchy tree in the Java development environment.
  - Every class in Java is a descendent (direct or indirect) of the Object class.
  - The Object class defines the basic state and behavior that all objects must have, such as
    - the ability to compare oneself to another object (equals())
    - to convert to a string (toString())
    - to return the object's class (getClass())

4.9 Inheritance - **Principles**

- Inheritance allows a software developer to derive a new class from an existing one.

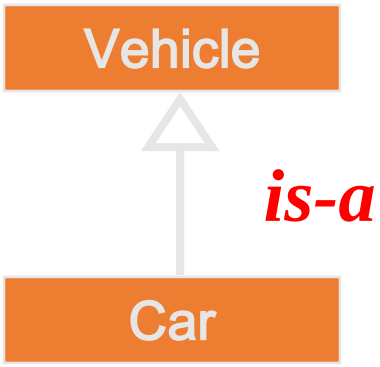
- The existing class is called the *parent* class, or *superclass,* or *base class*.
- The derived class is called the *child* class or *subclass*.

- As the name implies, the child inherits characteristics of the parent.
  - the child class inherits the methods and data/members defined for the parent class.

- To tailor a derived class, the programmer can add new members/variables or methods, or can modify the inherited ones.

4.9 Inheritance - **Graphical Representation**

- Inheritance relationships often are shown graphically in a UML class diagram, with an arrow with an open arrowhead pointing to the parent class

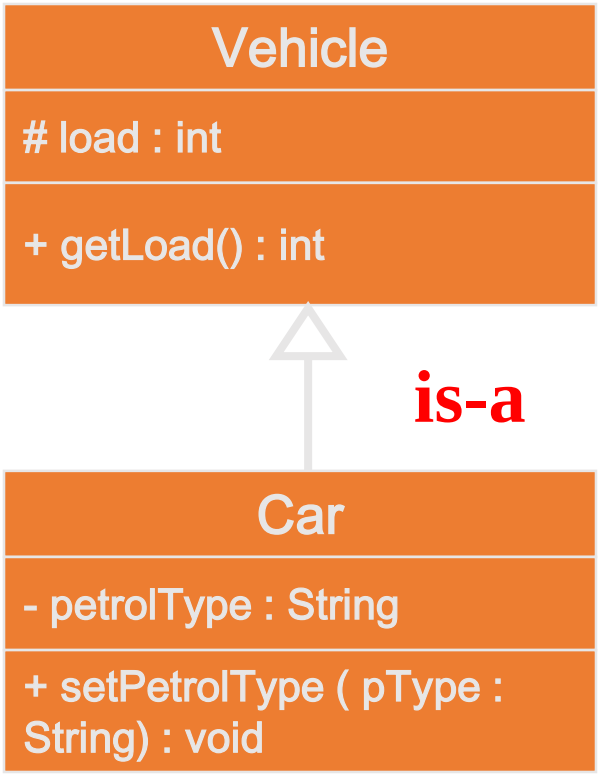


Inheritance should create an *is-a relationship*, meaning the child *is a* more specific version of the parent

4.9 Inheritance - **Graphical Representation**

- Visibility Modifiers:
  - public, protected, private

- UML diagram's notations:
  - "#" - Protected variables and methods
  - "-" - Private variables and methods
  - "+" - Public variables and methods

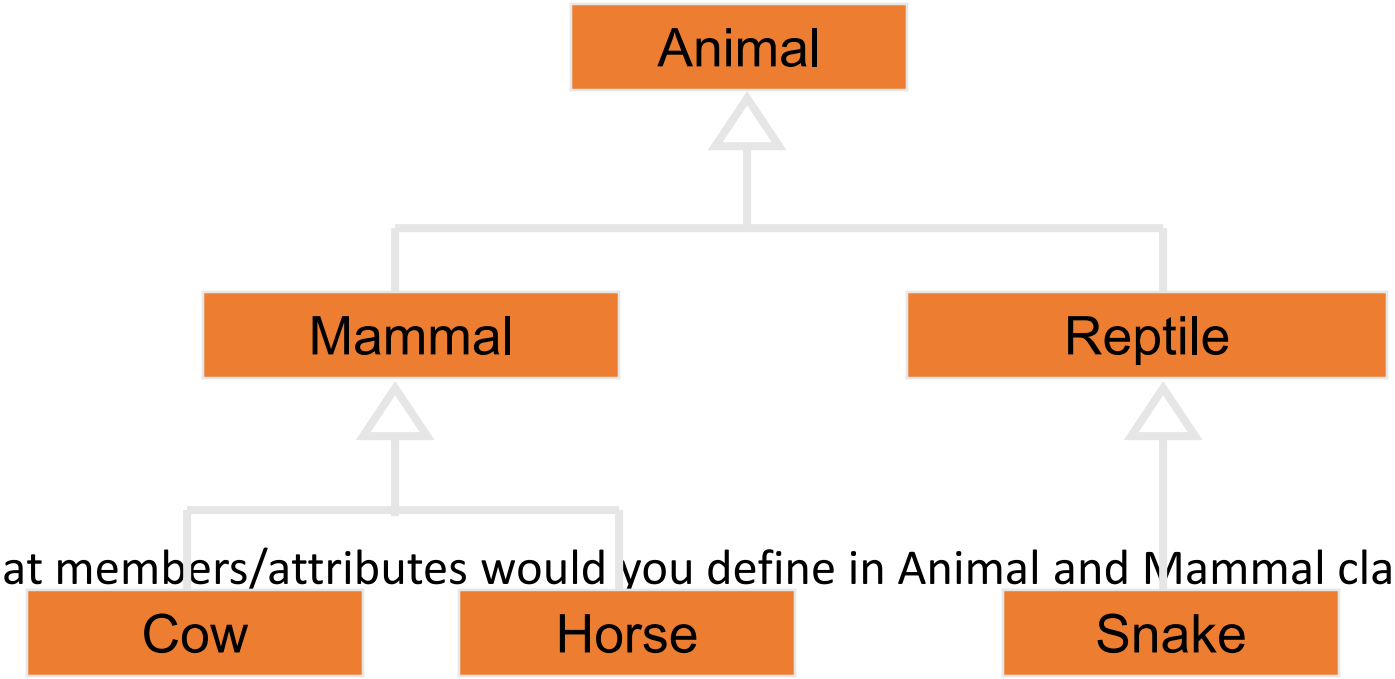


4.9 **Inheritance** - What is inherited?

- The following list itemizes the members that are inherited by a subclass:
  - **Subclasses inherit those superclass members declared as public or protected.**
  - **Subclasses don't inherit a superclass's member if the subclass declares a member with the same name.**
    - In the case of member variables, the member variable in the subclass **hides** the one in the superclass.
    - In the case of member methods, the method in the subclass **overrides** the one in the superclass.

4.9 Inheritance – **Another example**

- A child class of one parent can be the parent of another child, forming a *class hierarchy*



- What members/attributes would you define in Animal and Mammal classes ?

4.9 Inheritance - How we write it in Java?

- In Java, we use the reserved word `extends` to establish an inheritance relationship

```
class Car extends Vehicle
{
    // class contents
}
```

4.9 Inheritance – **Building the Class Hierarchies**

- Two children of the same parent are called *siblings*
- Common features should be put as high in the hierarchy as is reasonable
- An inherited member is passed continually down the line
- Therefore, a child class inherits from all its ancestor classes

4.9 Inheritance – Multiple Inheritance

- **Interface - a java alternative for multiple inheritance**
  - a Java class can implement multiple interfaces.
- **Have you ever found yourself wanting to write something similar to:**

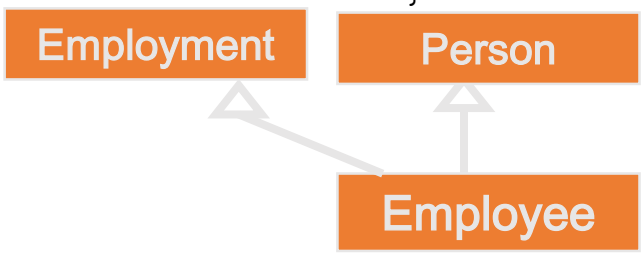
```
public class Employee extends Person, Employment
{
        // your code
}
```

Not correct!

  - Assume that you have already created

```
public class Person {
//code here
}
```

```
public class Employment {
//code here
}
```



4.9 Inheritance – Multiple Inheritance

- **The correct implementation that supports multiple inheritance is:**

```
public class Employee extends PersonInterf, EmploymentInterf
        {
                // your code
        }
```

Assume that you have already created

```
public interface PersonInterf {
  //code here
}

public class Person implements
PersonInterface {
  //code here
}
```

```
public interface EmploymentInterf {
//code here
}
public class Employment  implements
EmplymentInterf {
//code here
}
```

4.10 Stack Implementation

Class definition and constructor

```
import java.util.*;

public class MyStack implements StackInterface
{

    public ArrayList<String> theStack;

    /** Creates a new instance of Stack */
    public MyStack()
    {
        theStack = new ArrayList<String>();
    }
```

4.10 Stack Implementation

Stack Methods

```
  public boolean isEmpty()
  {
        return theStack.isEmpty();
  }
```

```
   /* isFull() is always false as there is no limit on  the
size of this arraylist based stack    */
 public boolean isFull()
 {
        return false;
 }
```

4.10 Stack Implementation

Stack Methods

```
/** puts an item onto the top of the stack */
 public void push(Object newItem) {
        theStack.add(0,(String)newItem);

}

/** removes the item  from the top of the stack and returns
it */
 public Object pop()
{
        if (!(theStack.isEmpty())) {
              return theStack.remove(0);
        }
        else
              return null;
 }
```

Casting down:
The type of the objects stored by the stack

4.10 Stack Implementation

Stack Methods

- **Other methods may also be implemented in the MyStack class in order to extend the functionality**
  - emptyStack() – removes ALL items that exist in the stack. Use pop() to remove one item at a time.
  - displayStack() – lists the content of the stack by parsing the elements. Returns a string with all the items from the stack.

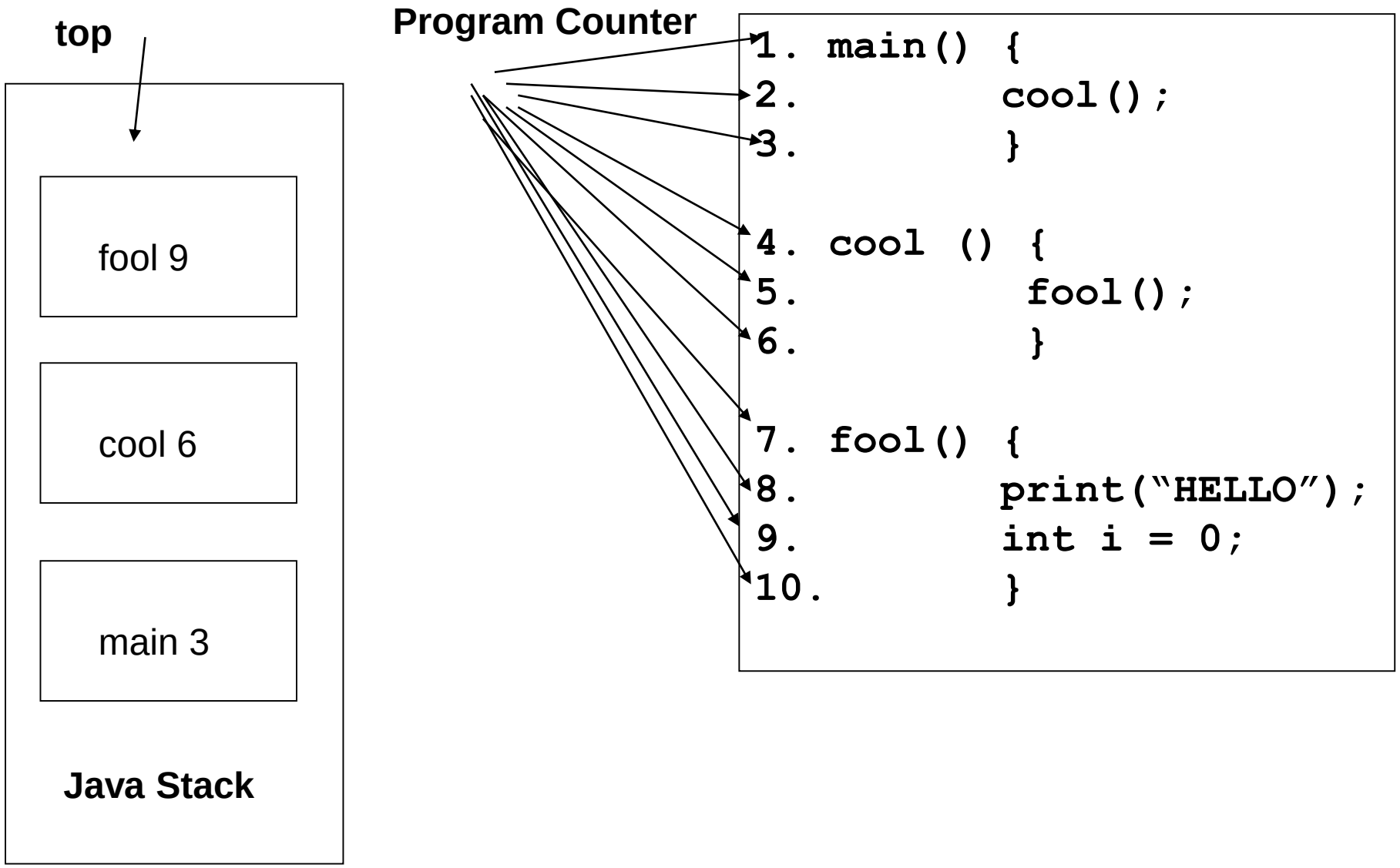
- **Printing/Parsing the stack elements:**
  - toString() method may be used directly on the MyStack object
    - You may re-write/implement (override) toString() method in the MyStack class
  - Iterator or FOR loop may be used to parse elements

4.11 Stack Application: JVM

- Stacks in the Java Virtual Machine
  - A Java program is typically compiled into a sequence of byte codes that are defined as “machine” instructions for the Java Virtual Machine (JVM).
  - The definition of the Java Virtual Machine is at the heart of the Java language itself. The stack data structure plays a central role in the definition of the Java Virtual Machine.
  - During the execution of a program , the Java Virtual Machine maintains a stack of descriptors of the currently active methods and their next instruction.
  - The JVM keeps in a special register called the program counter the address of the statement currently being executed in the program.

4.11 Stack Application: JVM

Principle



4.11 Stack Application: JVM

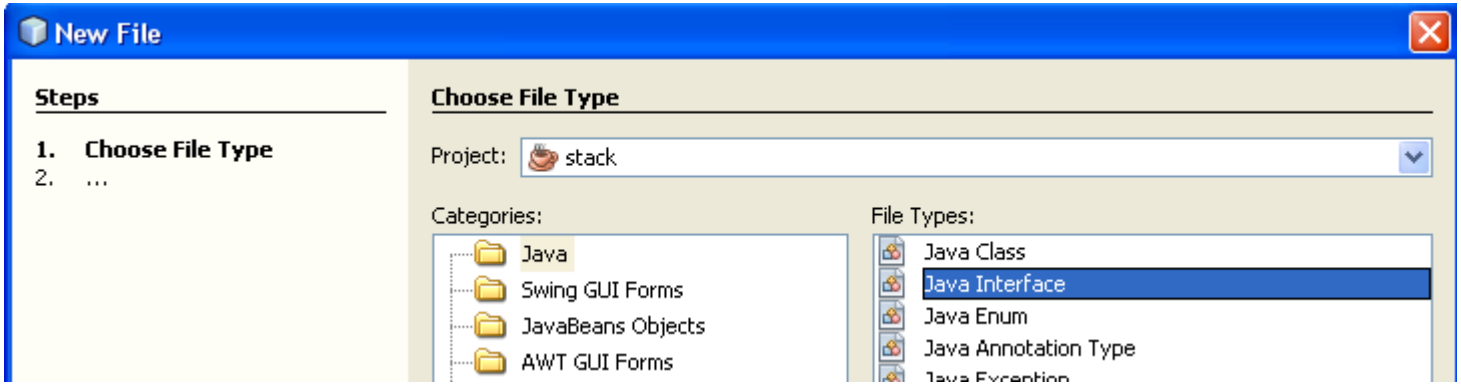
Example

Download JVM application

Look over the code and try to understand the order how messages are displayed

4.12 LAB: Application using MyStack ADT

- **Develop a NetBeans project that uses the stack ADT**
  - Create a project (called *stack)*
  - Name the main class (as *StackTester)*
  - Add a new Java file (called *StackInterface*) to the project that implement s the stack interface class and write the code for it
    - File -> New File and select Java Interface
  - Add a new Java file (*MyStack*) to the project that implements the stack class and write the code for it
    - File -> New File and select Java Class



4.12 LAB: Application using MyStack ADT

- **Write code in the main() method that uses the stack ADT.**
- **Common operations on the stack that involve calling methods implemented by your stack class (MyStack.java):**
  - Create a stack type object representing the stack
  - push one or more items into the stack (e.g. String type objects)
  - check if the stack is empty
  - Pop one item + print it out
  - Fully empty the stack (pop any remaining items)
  - Pop an item from an empty stack to see what happens

- **Your main() method will start with this…**

```
StackInterface si = new MyStack();
…
System.out.println("Stack is empty: " + si.isEmpty();
```

Summary

- Stack ADT implements LIFO principle
- Basic operations:
  - push, pop, isEmpty, isFull
- Interfaces
- Course web page:
  - **`StackInterface.java`**
  - **`MyStack.java`**
  - **`AnyClass.java`**