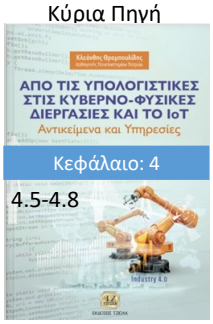
Αντικειμενοστρεφής Προγραμματισμός (Object-Oriented Programming)

(CEID_NNY106)

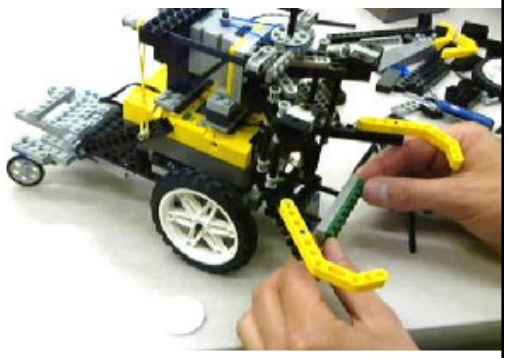
Η ΠΡΟΣΕΓΓΙΣΗ LEGO

Παίζοντας με τα "τουβλάκια" – Part B

Κύρια Πηγή



Μια **εισαγωγή στις βασικές έννοιες της ανάπτυξης προγράμματος** με βάση το αντικειμενοστρεφές παράδειγμα προγραμματισμού χρησιμοποιώντας την προσέγγιση Lego.



Kleanthis Thramboulidis
Prof. of Software and System Engineering
University of Patras
https://sites.google.com/site/thramboulidiskleanthis/

Η Άσκηση – Το πρόγραμμα Calc

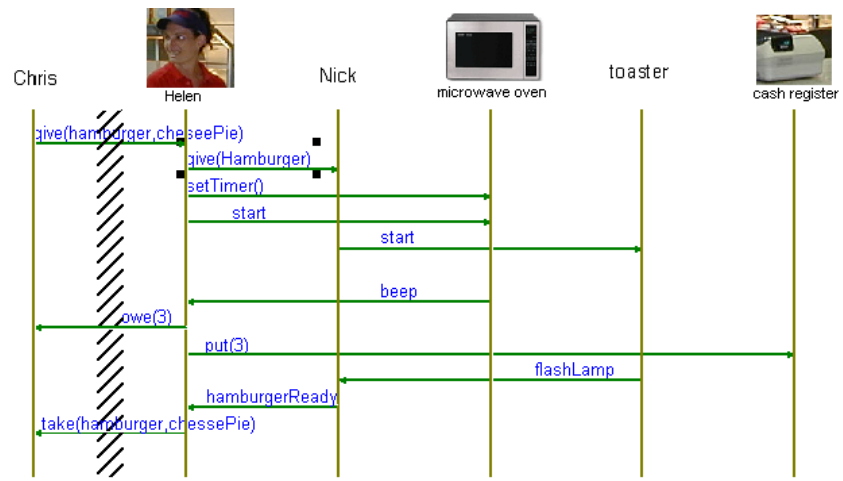
Δραστηριότητα 4.3
Calc App

Γράψτε ένα πρόγραμμα σύμφωνα με το οποίο το σύστημα θα δημιουργεί δύο στιγμιότυπα Double, θα τα βάζει στη στοίβα, στη συνέχεια θα τα παίρνει θα τα προσθέτει και το αποτέλεσμα θα το βάζει στη στοίβα. Από εκεί θα το παίρνει και θα το εμφανίζει στον χρήστη.

Οργάνωση Διάλεξης

- **Αφαιρετική περιγραφή συμπεριφοράς**
  - **Object Interaction Diagram/Sequence Diagram**
- Η άσκηση Calc
  - Αναγνώριση αντικειμένων (δομή προγράμματος)
- Το πρόγραμμα Calc (1η έκδοση)
  - Generics – java.util.Stack<E>
  - Η μέθοδος toString
  - System.out.println()
- Αύξηση τμηματοποίησης (modularity)

Object Interaction Diagram/Sequence Diagram



Περιγράφει συμπεριφορά συστήματος

"The whole is more than the sum of its parts" – Aristotle.

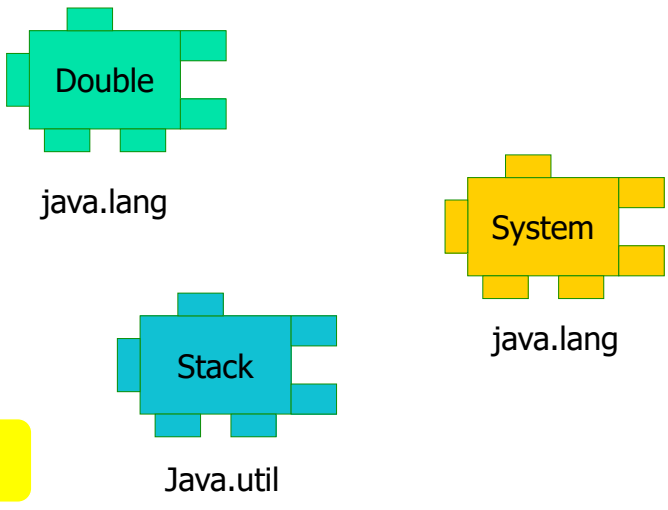
Οργάνωση Διάλεξης

- Αφαιρετική περιγραφή συμπεριφοράς
  - Object Interaction Diagram/Sequence Diagram
- **Η άσκηση Calc**
  - **Αναγνώριση αντικειμένων (δομή προγράμματος)**
- Το πρόγραμμα Calc (1η έκδοση)
  - Generics – java.util.Stack<E>
  - Η μέθοδος toString
  - System.out.println()
- Αύξηση τμηματοποίησης (modularity)

Calc - The constituent components

*Γράψτε ένα πρόγραμμα σύμφωνα με το οποίο το σύστημα θα δημιουργεί δύο στιγμιότυπα **Double**, θα τα βάζει στη στοίβα (**Stack**), στη συνέχεια θα τα παίρνει θα τα προσθέτει και το αποτέλεσμα θα το βάζει στη στοίβα. Από εκεί θα το παίρνει και θα το εμφανίζει στον χρήστη (**System**.out.println()).*

Edit-time vs. run-time

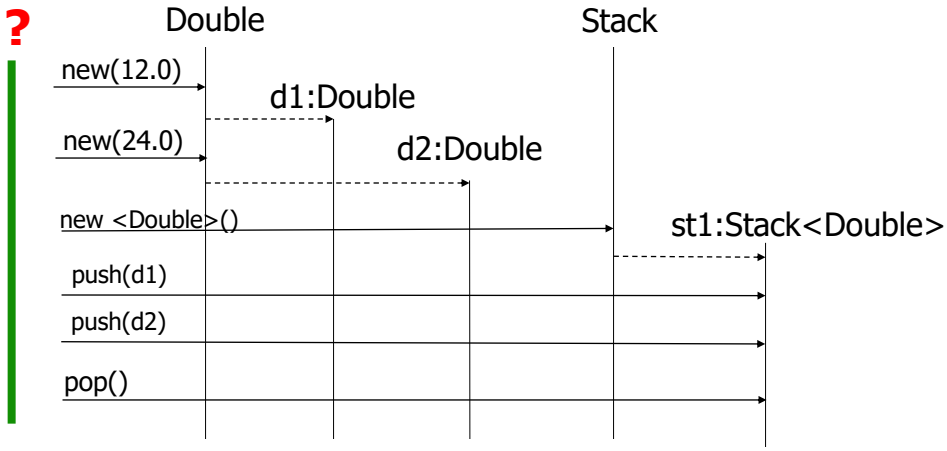


Την διεργασία εκτελέσαμε στο BlueJ (Ch4 PartA)

Calc-Διάγραμμα Αλληλεπίδρασης Αντικειμένων

UML sequence diagram

Το πρόγραμμα …
*θα δημιουργεί δύο στιγμιότυπα **Double**, θα τα βάζει στη στοίβα (**Stack**), στη συνέχεια θα τα παίρνει θα τα προσθέτει και το αποτέλεσμα θα το βάζει στη στοίβα. Από εκεί θα το παίρνει και θα το εμφανίζει στον χρήστη (**System**.out.println()).*

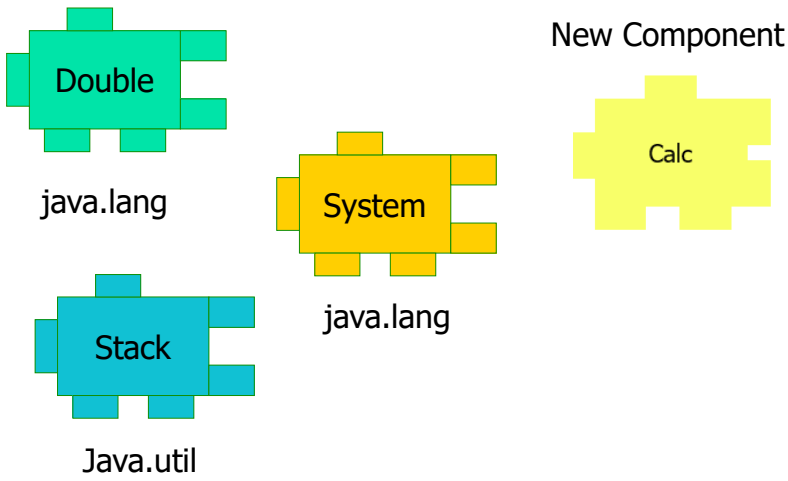


Ανάγκη για δημιουργία νέου αντικειμένου

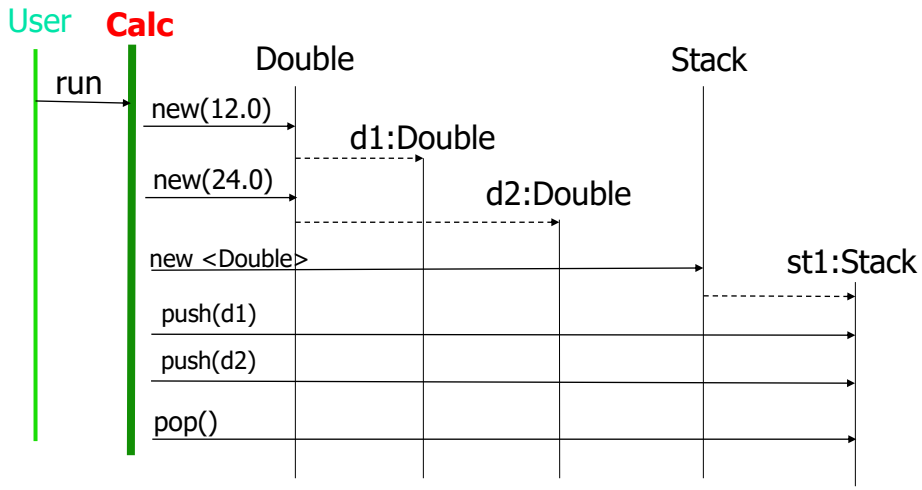
Calc - The constituent components

Edit-time vs. run-time

*Γράψτε ένα **πρόγραμμα** σύμφωνα με το οποίο το σύστημα θα δημιουργεί δύο στιγμιότυπα **Double**, θα τα βάζει στη στοίβα (**Stack**), στη συνέχεια θα τα παίρνει θα τα προσθέτει και το αποτέλεσμα θα το βάζει στη στοίβα. Από εκεί θα το παίρνει και θα το εμφανίζει στον χρήστη (**System**.out.println()).*



Διάγραμμα Αλληλεπίδρασης Αντικειμένων



Ορισμός της κλάσης Calc

?

```
class Calc {
    //data members ?
    // methods ?
    }
```

```
class Calc {
    static Double d1, d2;
    static Stack<Double> st1;

    public static void main() {
        ?
        }
    }
```

Οργάνωση Διάλεξης

- Αφαιρετική περιγραφή συμπεριφοράς
  - Object Interaction Diagram/Sequence Diagram
- Η άσκηση Calc
  - Αναγνώριση αντικειμένων (δομή προγράμματος)
- **Το πρόγραμμα Calc (1η έκδοση)**
  - **Generics – java.util.Stack<E>**
  - **Η μέθοδος toString**
  - **System.out.println()**
- Αύξηση τμηματοποίησης (modularity)

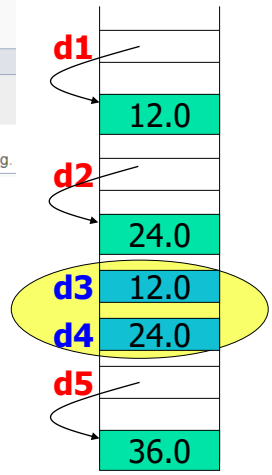
Δημιουργία στιγμιότυπων της Double

Constructor Summary

Constructors

Constructor and Description
`Double(double value)` Constructs a newly allocated `Double` object that represents the primitive `double` argument.
`Double(String s)` Constructs a newly allocated `Double` object that represents the floating-point value of type `double` represented by the string.

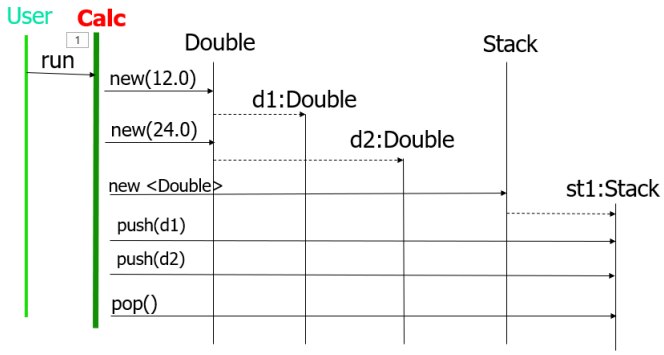
```
Double d1= new Double(12.0);
Double d2 = new Double("24.0");
double d3 = d1.doubleValue();
double d4 = d2.doubleValue();
Double d5 = new Double(d3+d4);
```



double	**doubleValue()** Returns the double value of this Double object.

class Calc (1st version)

```
public class Calc{
    static Double d1,d2;
    static  Stack<Double> st;

    public static void main(){
         d1 = new Double(20);
          d2 = new Double(40);
         st = new Stack<Double>();
         st.push(d1);
         st.push(d2);
         Double d3 = st.pop();
         Double d4 = st.pop();
         st.push(d3+d4);
         System.out.println("sum is="+ st.pop());
    }
```



Compare it with the object interaction diagram

Generics …. Stack<E>

```
public class Stack<E>
extends Vector<E>
```

- enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods.
- Much like the more familiar formal parameters used in method declarations, **type parameters** provide a way for you to re-use the same code with different inputs. The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types.
- Code that uses generics has many benefits over non-generic code:
  - Stronger type checks at compile time.
    A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.
- Elimination of casts.
  - Enabling programmers to implement generic algorithms.
    By using generics, programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read.

The first program in Java – System.out.println

```
/**************************************************************
* File name : HelloWorld.java
* Compile with javac:        javac HelloWorld.java
* Execute with java:         java HelloWorld
*
* Developer's Comments
* The HelloWorld class implements an application that simply
* displays "Hello World!" to the standard output.
**************************************************************/

public class HelloWorld {

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

    }
```

Βοοκ: Ενότητα 4.6

Η κλάση System

```
public final class
  java.lang.System extends
  java.lang.Object  {
      // Fields
  public static PrintStream err;
  public static InputStream in;
  public static PrintStream out;
```

Δες Βασική βιβλιοθήκη της Java

out
The "standard" output stream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output or another output destination specified by the host environment or user.

Η κλάση PrintStream – method overloading

```
public class java.io.PrintStream
    extends java.io.FilterOutputStream {
    // Methods
    public void print(boolean b);
    …
    public void println();
    public void println(boolean b);
    public void println(char c);
    public void println(char s[]);
    public void println(double d);
    public void println(float f);
    public void println(int i);
    public void println(long l);
    public void println(Object obj);
    public void println(String s);
    …
   }
```

Η μέθοδος toString

Δραστηριότητα 4.4
System.out.println("Stack status: " + st)

Κάντε μια προσπάθεια να αναγνωρίσετε ποια από τις println() της PrintStream καλεί το σύστημα με βάση την παρακάτω πρόταση της main()

```
System.out.println("Stack status: " + st);
```

```
10 public class Calc{
11     static Double d1,d2;
12     static  Stack<Double> st;
13
14     public static void main(){
15         d1 = new Double(20);
16         d2 = new Double(40);
17         st = new Stack<Double>();
18         st.push(d1);
19         st.push(d2);
20         System.out.println("Stack status: "+st);
```

toString

```
public String toString()
```

Returns a string representation of this Vector, containing the String representation of each element.

BlueJ: Terminal Window - CalcV1

Options

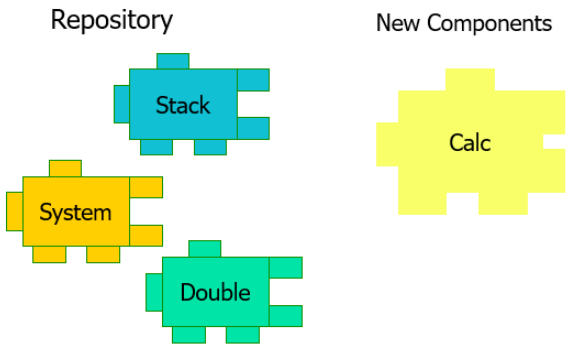
```
Stack status: [20.0, 40.0]
sum is=60.0
```

Οργάνωση Διάλεξης

- Αφαιρετική περιγραφή συμπεριφοράς
  - Object Interaction Diagram/Sequence Diagram
- Η άσκηση Calc
  - Αναγνώριση αντικειμένων (δομή προγράμματος)
- Το πρόγραμμα Calc (1η έκδοση)
  - Generics – java.util.Stack<E>
  - Η μέθοδος toString
  - System.out.println()
- **Αύξηση τμηματοποίησης (modularity)**

2nd iteration - Increase reusability (Χρήση μεθόδου)

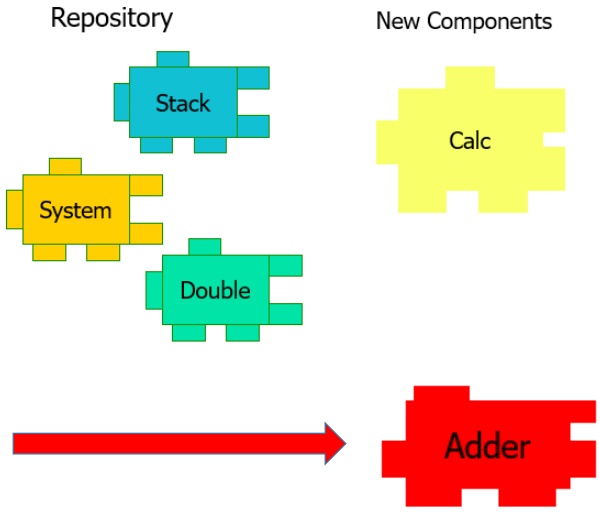
```
public class Calc{
   ....
   public static void main(){
        d1 = new Double(20);
        d2 = new Double(40);

        st = new Stack<Double>();
        st.push(d1);
        st.push(d2);

        Double d3 = st.pop();
        Double d4 = st.pop();
        st.push(d3+d4);
        System.out.println("sum is="+st.pop());
   }
```



Method add()

2nd iteration - Increase reusability (Χρήση αντικειμενου)

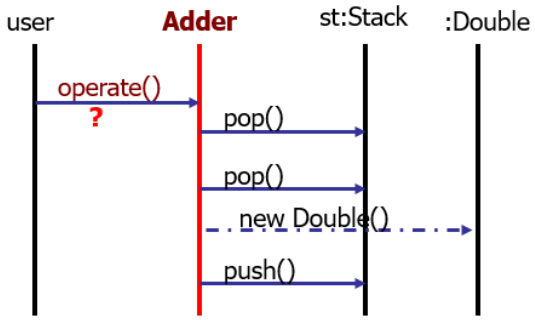
```
public class Calc{
    ....
    public static void main(){
        d1 = new Double(20);
        d2 = new Double(40);

        st = new Stack<Double>();
        st.push(d1);
        st.push(d2);

        Double d3 = st.pop();
        Double d4 = st.pop();
        st.push(d3+d4);
        System.out.println("sum is="+st.pop());
    }
```



How do they communicate for the program to provide the plus functionality?

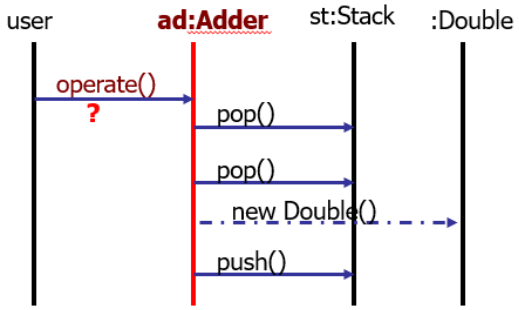
Build new Component – Μέθοδος κλάσης


```
class Adder{
    public static void operate(){
        Double d3 = st.pop();
        Double d4 = st.pop();
        st.push(new
Double(d3+d4));
    }
}
```



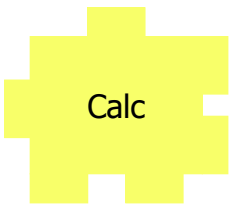
Build new Component – Μέθοδος στιγμιότυπου


```
class Adder{
    public void operate(){
        Double d3 = st.pop();
        Double d4 = st.pop();
        st.push(new Double(d3+d4));
    }
}
```



Integrate components

```
public class Calc{
    public static void main(){
        Double d1 = new Double(20);
        System.out.println("d1="+d1);
        Double d2 = new Double(40);

        Stack<Double> st = new Stack<Double>();
        st.push(d1);
        st.push(d2);
        ......
        System.out.println("sum is="+st.pop());
    }
```



?


```
class Adder{
    public static void operate(){
        Double d3 = st.pop();
        Double d4 = st.pop();
        st.push(new Double(d3+d4));
    }
}
```

Flow of control ?

?

Source code (2nd version)

```
public class Calc{
    public static Stack<Double> st;
    public static void main(){
        Double d1 = new Double(20);
        System.out.println("d1="+d1);
        Double d2 = new Double(40);

        st = new Stack<Double>();
        st.push(d1);
        st.push(d2);

        Adder.operate();
        System.out.println("sum is="+st.pop());
    }
```

?

```
class Adder{
    public static void operate(){
        Double d3 = Calc.st.pop();
        Double d4 = Calc.st.pop();
        st.push(new Double(d1+d2));
    }
}
```

Identify alternative Implementations for addition

Εναλλακτικές υλοποιήσεις

Kleanthis Thramboulidis
Διαχειριστής · 25 Οκτωβρίου στις 5:03 μ.μ. · Exercise

Εναλλακτικές Υλοποιήσεις #Ασκηση #RPNCalculator #Reusabililty #Activity3

Το 2ο iteration της δραστηριότητας 3 έχει ως στόχο την αύξηση του modularity του κώδικα του 1ου iteration.

Επιλέξαμε το παρακάτω τμήμα τους κώδικα της main που υλοποιεί μια συγκεκριμένη λειτουργικότητα (functionality).

Double d3 = st.pop();

Double d4 = st.pop();

st.push(d3+d4);

Το iteration αυτό έχει ως στόχο την αναζήτηση εναλλακτικών υλοποιήσεων που θα μας επιτρέπουν την επαναχρησιμοποίηση της λειτουργικότητας αυτής.

Δώστε με απάντηση στην ανάρτηση αυτή μια πρόταση υλοποίησης αφού πρώτα έχετε ελέγξει πως αυτή δεν έχει περιγραφεί από κάποιον άλλον πριν από εσάς.

Περιγράψτε με λόγια την προτεινόμενη υλοποίηση και τον τρόπο που η λειτουργικότητα αυτή μπορεί να αξιοποιηθεί στα πλαίσια της main.

Δώστε στο τέλος της περιγραφής σας το URL του pdf αρχείου με τον κώδικα

Αφαιρετική αναπαράστασης Δομής – UML class diagram