Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Teaching OOP

&

A Practical Approach Using BlueJ

David J Barnes
University of Canterbury

Michael Kölling
University of Southern Denmark

Introduction

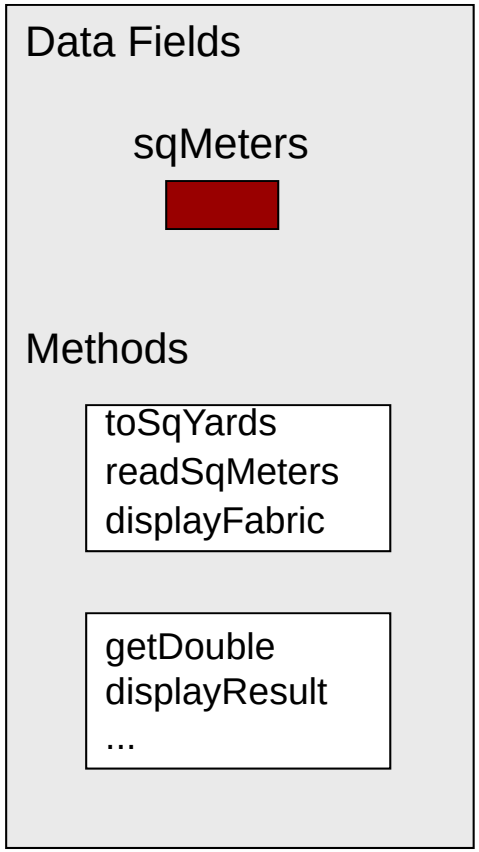
- Who are you?
- What/where do you teach?
- What is your experience with Java teaching?
- What do you expect from this workshop?
- What problems do you experience in your teaching?

Analysis

What is happening with OOP teaching?

Functions

Conversion: square metres to square yards for a piece of fabric



Hello World

```
public class Hello
{
   public static void main(String[] args)
   {
     System.out.println(“Hello, world”);
   }
}
```

Text

```
public class OldMacDonald {
    public static void main(String[] args) {
        System.out.println(“Old MacDonald had a farm”);
        System.out.println(“E I E I O”);
        System.out.println(“and on his farm he had a duck”);
        System.out.println(“E I E I O”);
        System.out.println(“With a quak quak here”);
        System.out.println(“And a quak quak there”);
        System.out.println(“Here a quak, there a quak”);
        System.out.println(“Everywhere a quak quak”);
        System.out.println(“Old MacDonald had a farm”);
        System.out.println(“E I E I O”);
}  }
```

1. Write a Java class that prints the second verse
2. Write a Java class that prints the following design...

GUI's

```
import java.awt.*;
public class Message2 extends Frame {
  Font f;
  public Message2() {
    f = new Font("SansSerif", Font.BOLD, 24);
    setBackground(Color.yellow);
    setSize(400, 150);
  }
  public void paint(Graphics g) {
    g.setFont(f);  g.setColor(Color.blue);
    g.drawString("Welcome to Java", 100, 100);
  }
  public static void main(String[] args) {
    Message2 m2 = new Message2();
    m2.setVisible(true);
  }
}
```

Text input

```
import java.io.*;

public class Square
{
    public static void main(String[] args) throws Exception
    {
        BufferedReader reader =
            new BufferedReader(
                new InputStreamReader(System.in));
        System.out.println("Input a number");
        int number = Integer.parseInt(reader.readline());
        System.out.println(number + " squared is " +
                           (number * number));
    }
}
```

Why are things like this?

- It has a lot to do with the Pascal (etc.) heritage.
- Just using a class doesn't make an OO program.
- Using an OO language requires a proper OO approach.

Getting started

How to get over the first few weeks

Suggestions #1

- Objects first
- Jump start
- Don’t use “main”
- Don’t start with I/O (GUIs, applets, text I/O)
- Don’t convert old examples
- Teach programming, not a programming language

Objects first

- Start with key issues:
  Objects, classes, methods
- State and behaviour
- Almost universally accepted, but: not easy

Jump start

- Don’t get students bored with detail at the start
- Start doing something!
- Experiment with objects before writing objects.

Don’t use “main”

- What has main got to do with objects? - Nothing!

```
public class Hello
{
   public static void main(String[] args)
   {
     System.out.println(“Hello, world”);
   }
}
```

Don’t start with I/O

- I/O is an undesirable distraction from programming principles
- Text I/O (especially input) is not easy
- Applets are mysterious
- GUIs are complicated
- Custom made libraries…?

Don’t convert your old examples

- Examples are (or should be) handpicked to illustrate key points
- Examples from procedural languages do not illustrate concepts of object orientation

Teach programming, not a programming language

- Don’t teach Java, teach OOP!
- Don’t teach OOP, teach programming!
- Don’t teach programming, teach problem solving!

Environment again

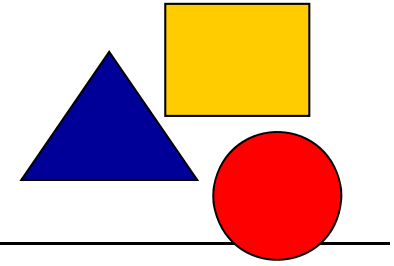
- need visualisation
- need interaction
- need simplicity
- BlueJ
  - successor of Blue system
  - supports Java
  - written in Java
  - free

An Object-Oriented Environment

- An environment for an object-oriented language is not the same as an object-oriented environment

Examples

Shapes and Pictures



Questions / Discussion

Break

Conveying key topics

Examples

Ticket machines &
technical support

Suggestions #2

- Key issues first
- Read code
- Use “large” projects
- Don’t start with a blank screen
- Do sensible things
- Prepare for maintenance

Key issues first

- Choose examples to illustrate key issues first
- Don't get stuck in detail
- Avoid completeness trap!

Read code

- Make students read code! (read before write)
- Show many examples
- Make sure all examples are well written (and worth reading)

Use “large” projects

- Don’t start with small examples - they don’t have structure
- Use realistic context
- Concentrate on object interaction

Don’t start with a blank screen

- Starting with a blank screen requires design (or the example is too trivial)
- Design is hard
- Designing a complete application is an advanced exercise

Do sensible things

- Choose examples that make sense
- Don’t make students do things that you wouldn’t do yourself

Prepare for maintenance

- Prepare students for real-life situations:
  - code maintenance
  - modification, extension
  - documentation
  - using class libraries
  - reuse

Example

The dark world of Zuul…

Suggestions #3

- Discuss application structure
- Open/closed exercises
- Create ownership
- Apprentice approach

Discuss structure

- OO-modelling
- Relationships of classes
- Coupling and cohesion
- Later: patterns

Open/closed exercises

- Make exercises that are closed so students know exactly what is expected of them (and when they are done), and
- Make exercises that are open so each student can adapt them to his/her own level of knowledge, expertise and ambition

Create ownership

- Create your exercises and programming projects in such a way that the students take ownership
- Students then will
  - work harder
  - have more fun
  - learn much, much more…

Apprentice approach

- Let students observe what you are doing
- See teacher in action
- Learn by imitation

Reflection / Discussion

References

The material from this workshop will be available on the web:

`www.mip.sdu.dk/~mik/bluej-workshop`

David J Barnes, Michael Kölling
Objects First with Java -
A Practical Introduction using BlueJ

http://www.bluej.org

Contact: David Barnes (djb@kent.ac.uk)
Michael Kölling (mik@mip.sdu.dk)