Programming Workshop
Lecture 1

Carsten Schürmann
Alexandre Buisse

March 23, 2009

Motivation

"If you are the only one who can understand your own handiwork, it puts you in an eviable bargaining position."

Financial Times
March 19 2009
AIG reinvents the trader's option

Goals of the Programming Workshop

- Program a bigger project.
- Reflect about your design decision.
- Justify your design decision in written form (project report).
- Learn how to comment code.
- Learn how to improve the quality of your code.

What You May Expect From Us

People:
- Course Responsible: Carsten Schürmann (4C13)
- Teacher: Alexandre Buisse (4C16)
- Teaching Assistant: Mark Turski
- Office hours: lab hours and by appointment.

Grading:
- This course supplements GP and OOP.
- No exam.
- Weekly handing are mandatory.
- Final grade: 50% report + 50% code.

WWW:
- `http://www.itu.dk/courses/WP`
- `carsten|abui|mttu@itu.dk`

What We Expect From You

- You *must* write the report by yourself.
- You *must* write the code by yourself.
- Plagarizing, copying, and cheating is not tolerated.
- I am serious, and yes, I will be merciless.
- Discussing and brainstorming in groups is encouraged.
- Policy on extensions: Should you need an extension ask us before the deadline. Extensions petitioned after the deadline will be rejected.

Art of Programming

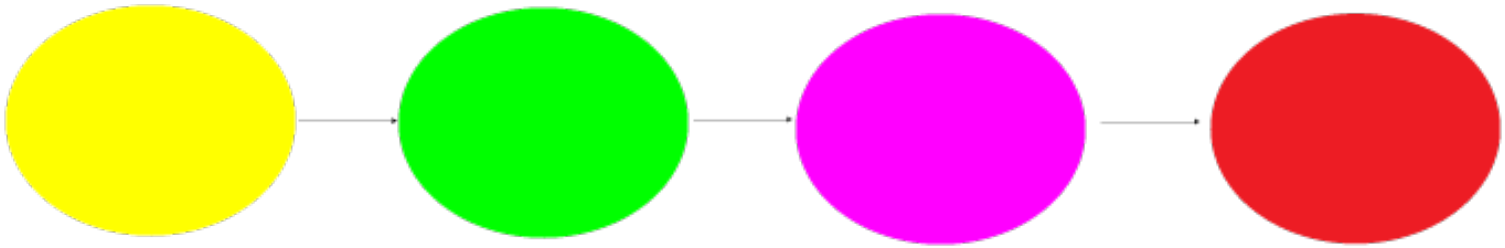
Consider

```
int function1 (A b) {
  return (b.c ());
}
```

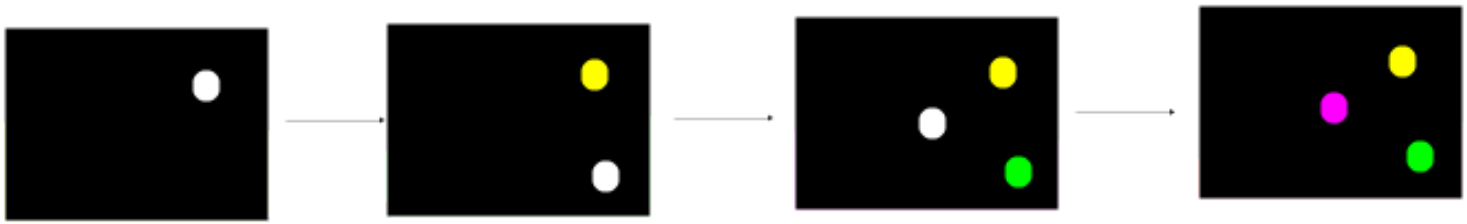
What does this piece of code do?

Effect-free vs. Effect-full

Effect-free (Local State, Method Invocation, Parameter Binding)



Effects (Computer Memory, Global State, Exceptions)



Specification

Write a program that takes a `String` of one or two digits. Write a program that returns how many digits were entered.

Mental model of a computing machine should comprise two parts

Effect-free
- Local parameters in a method.
- Return values.

Effect-full
- Computer Memory (Object references).

Definition (Specification)

A *specification* is a statment addressing the effect-free parts.

Definition (Invariant)

The property that remains (ideally) invariant during execution. A *class invariant* is a statment about the fields of an class. *state invariant* is a statement about the entire memory.

Partial Correctness

Define

- State/class invariants for your classes
- specifications for your methods

such that you can justify all specifications (line by line).

Comment to specify.

```
/*
Input:  a string of 1 or 2 digits.
Output: an integer number corresponding
        to the number of digits
*/
```

Partial Correctness [cont'd]

Comment to justify.

```
int function1 (String s) {
  return (s.length());
  /* by the specification of length in class String */
}
```

Do not comment for anything else.

What we want to see:

1. That you can specify!
2. That you can justify!

We will read you code, we will follow every step. Your code will be rejected if we cannot follow your train of thought.

Total Correctness

A partially correct program is *totally correct*,
if it always terminates.

```
class Fib {
  /* Input: An integer n
     Output: An integer fib (n) */
  int nth (int n) {
    if (n <= 1) {return 1;}
    else {return (nth (n-1) + nth (n-2));}
    /* Specification follows by the definition
       of Fibonacci numers  */
  }
  /* Input: Nothing
     Output: fib(5)  */
  public static void main (String[] args) {
    Fib f = new Fib ();
    System.out.println (f.nth (5));
  }
}
/* State Invariant: Only objects of Fib are in memory */
/* Class Invariant: NONE */
```

Extended Example

Specification

- There are persons.
- There are cars.
- Every car must be owned by a person.
- A car has at most one owner.

Extended Example (Car)

```
class Car {
  public Person owner;

  public Car(Person owner){
    setOwner(owner);
  }
  public void setOwner(Person newOwner){
    if (owner != null) owner.setCar(null);
    owner = newOwner;
    newOwner.setCar(this);
  }
  public Person getOwner(){
    return owner;
  }
}
```

Extended Example (Person)

```
class Person {
  public Car car;

  public Person (){
    this.car = NULL;
  }
  public void setCar(Car car){
    this.car = car;
  }
  public Car getCar(){
    return car;
  }
}
```

Specification for Car

```
/* State Invariant:
     Memory contains only Owners and Cars
     For all cars: car of the owner of a car is the car.
     For all persons: Either the car of an owner is NULL
            or owner of the car of a owner is the owner.
   Class Invariant: owner is not NULL  */

/* Specification for constructor Car
   Input: owner is not NULL.  Output: NONE
*/

/* Specification for method setOwner
   Input: newOwner is not NULL.  Output: NONE
*/

/* Specification for method getCar
   Input: NONE. Output: the owner of the car.
*/
```

Specification for Person

```
/* Class Invariant: NONE */

/* Specification for constructor Person
   Input, Output: NONE
*/

/* Specification for method setCar
   Input: car = NULL or car.owner = this
   Output: NONE
*/

/* Specification for method getCar
   Input: NONE
   Output: the car which is owned or NULL
*/
```

Good luck!

- Interfaces approximate specification.
- Only justify unclear reasoning.