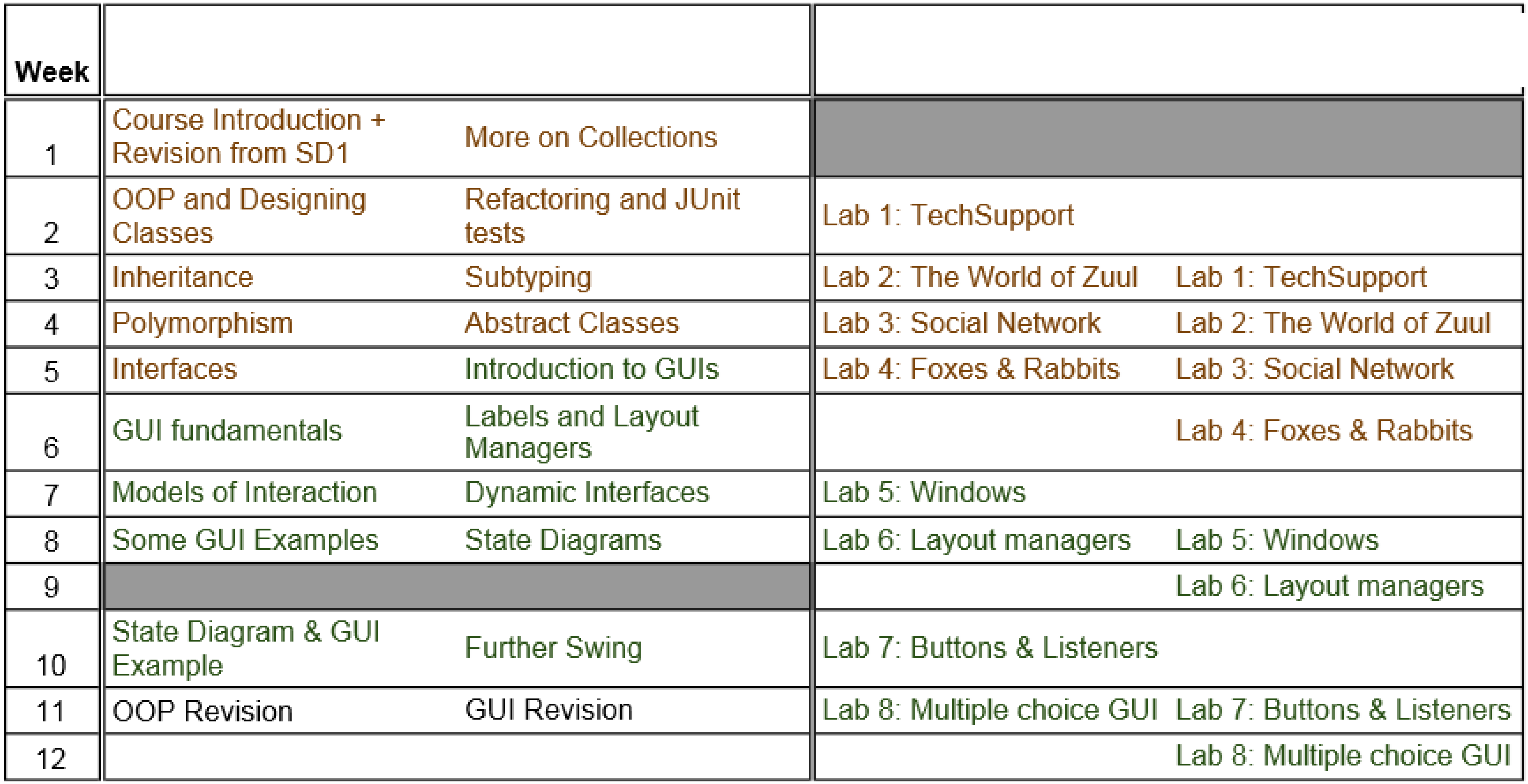
F27SB -Software Development 2

Schedule



Week				
1	Course Introduction + Revision from SD1	More on Collections		
2	OOP and Designing Classes	Refactoring and JUnit tests	Lab 1: TechSupport	
3	Inheritance	Subtyping	Lab 2: The World of Zuul	Lab 1: TechSupport
4	Polymorphism	Abstract Classes	Lab 3: Social Network	Lab 2: The World of Zuul
5	Interfaces	Introduction to GUIs	Lab 4: Foxes & Rabbits	Lab 3: Social Network
6	GUI fundamentals	Labels and Layout Managers		Lab 4: Foxes & Rabbits
7	Models of Interaction	Dynamic Interfaces	Lab 5: Windows	
8	Some GUI Examples	State Diagrams	Lab 6: Layout managers	Lab 5: Windows
9				Lab 6: Layout managers
10	State Diagram & GUI Example	Further Swing	Lab 7: Buttons & Listeners	
11	OOP Revision	GUI Revision	Lab 8: Multiple choice GUI	Lab 7: Buttons & Listeners
12				Lab 8: Multiple choice GUI

Assignment and Grading

- **50% final exam.**
- **50% code assignments:**
  - Programming exercises in Labs.
  - 1 week to complete.
  - Checked off by lab helpers.

Additional Learning Material

- Find us on VISION! (slides, code etc.)
  - Please check for regular updates.

- BlueJ Book: "Objects First with Java" by David J. Barnes & Michael Kölling.
  - Multiple copies are available from the **library**.
  - 4th, 5th edition.
  - Chapters 6-10

Objects First with Java
A Practical Introduction Using BlueJ
Fourth Edition
CD-ROM Included
David Barnes & Michael Kölling

- The (official) Java Online Tutorials.
  - http://docs.oracle.com/javase/tutorial/

Syllabus

Part 1 Object Oriented Programming

- **Inheritance and Generics:** hierarchies, subclasses, polymorphism, static and dynamic type, overriding, dynamic method lookup.
- **Designing classes:** coupling, cohesion, abstraction, modularisation, types
- **Abstract classes, abstract methods, interfaces**

•**Refactoring** and **JUnit tests**

Part 2

•State machines & state

diagrams

- GUIs: components, layout, event handling

Learning Outcomes

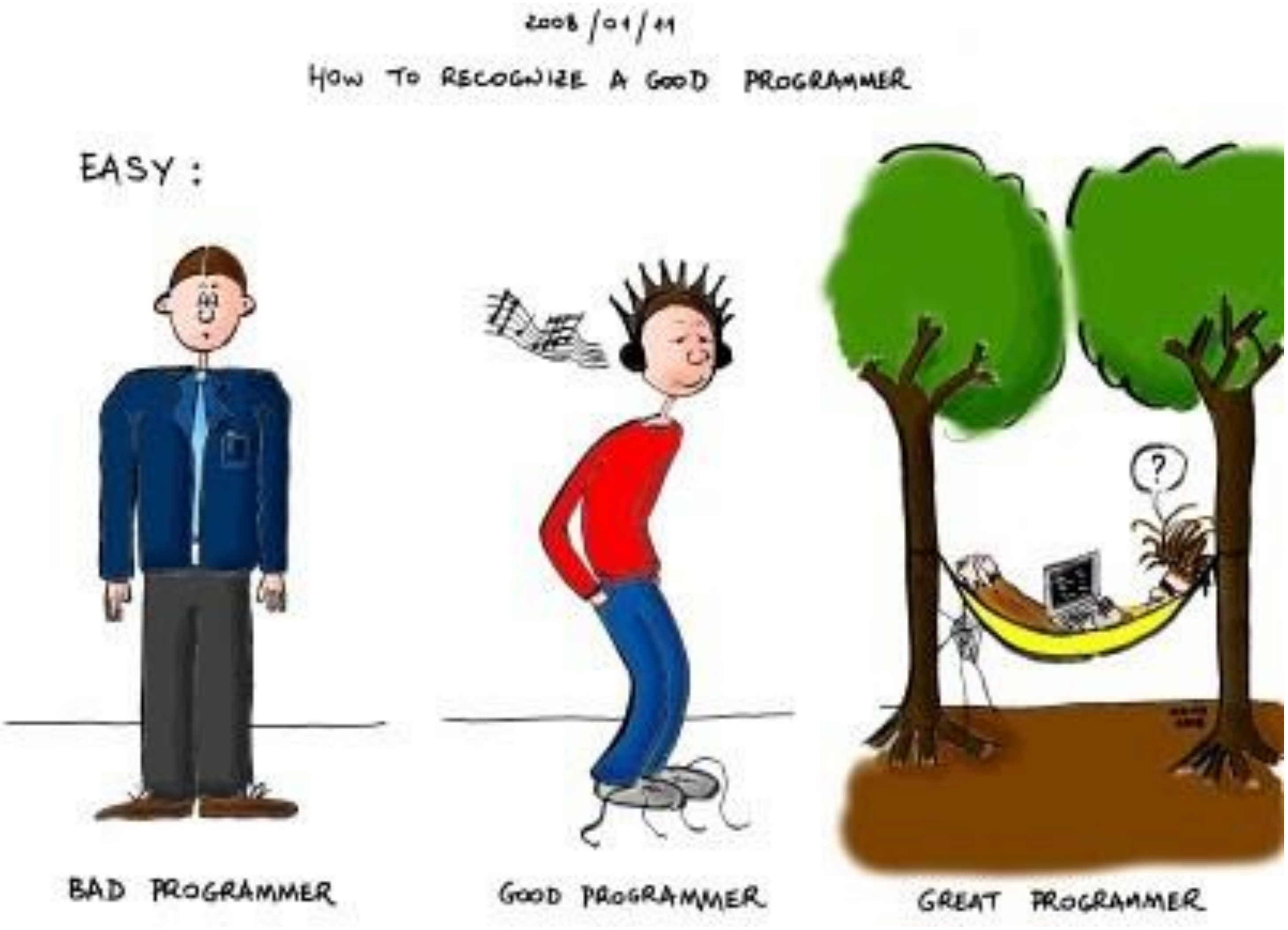
Aims of the first part of this course:

- Understand what’s “good” and what’s “bad” code.
- Be able to critically **evaluate** code.
- Apply **Object Oriented Programming (OOP)** principles to **improve** code.
- Make **design decisions** based on **OOP principles.**

Course Goals

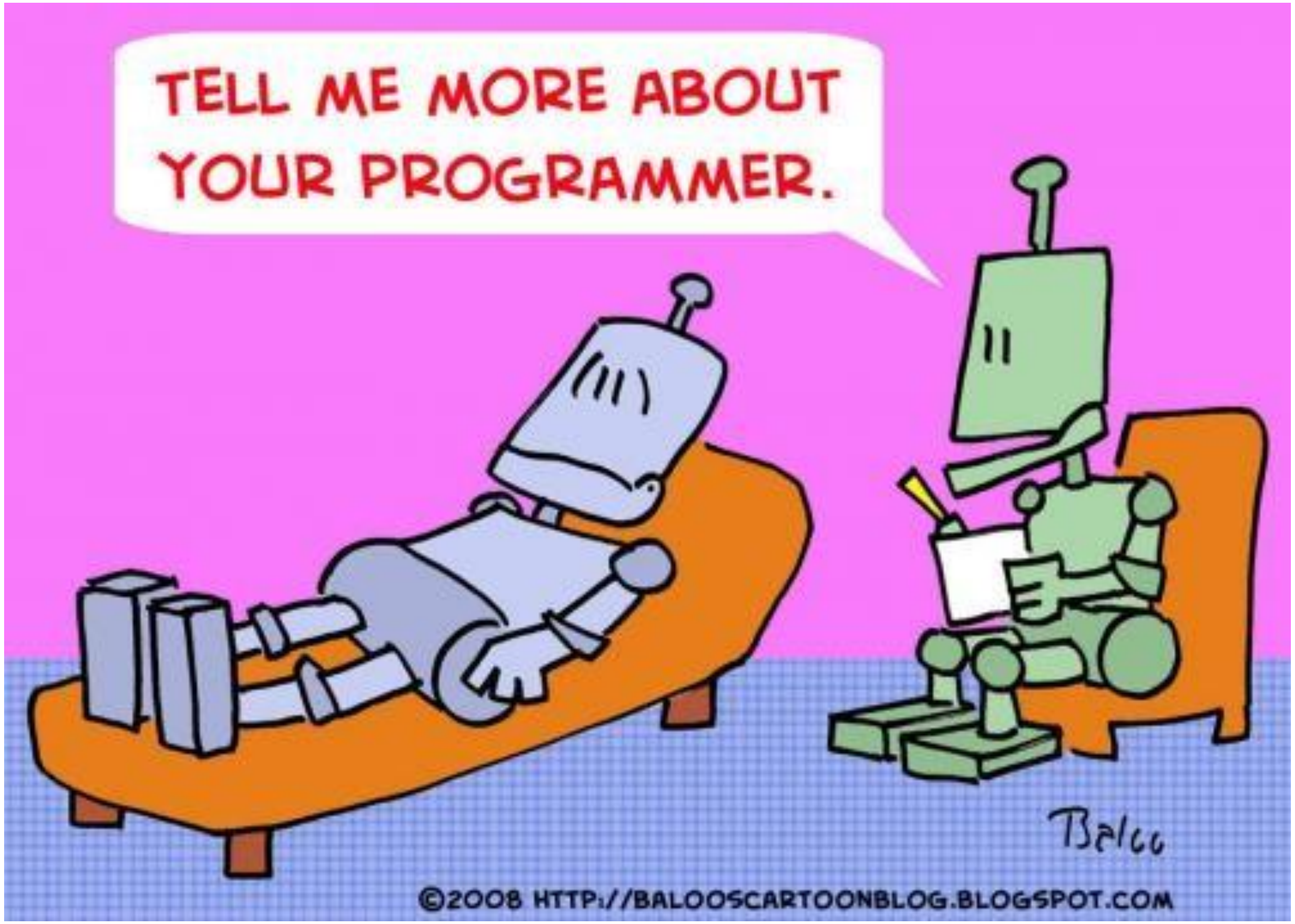
- By now you know how to write code which "does the job".

How to spot a goodprogrammer?

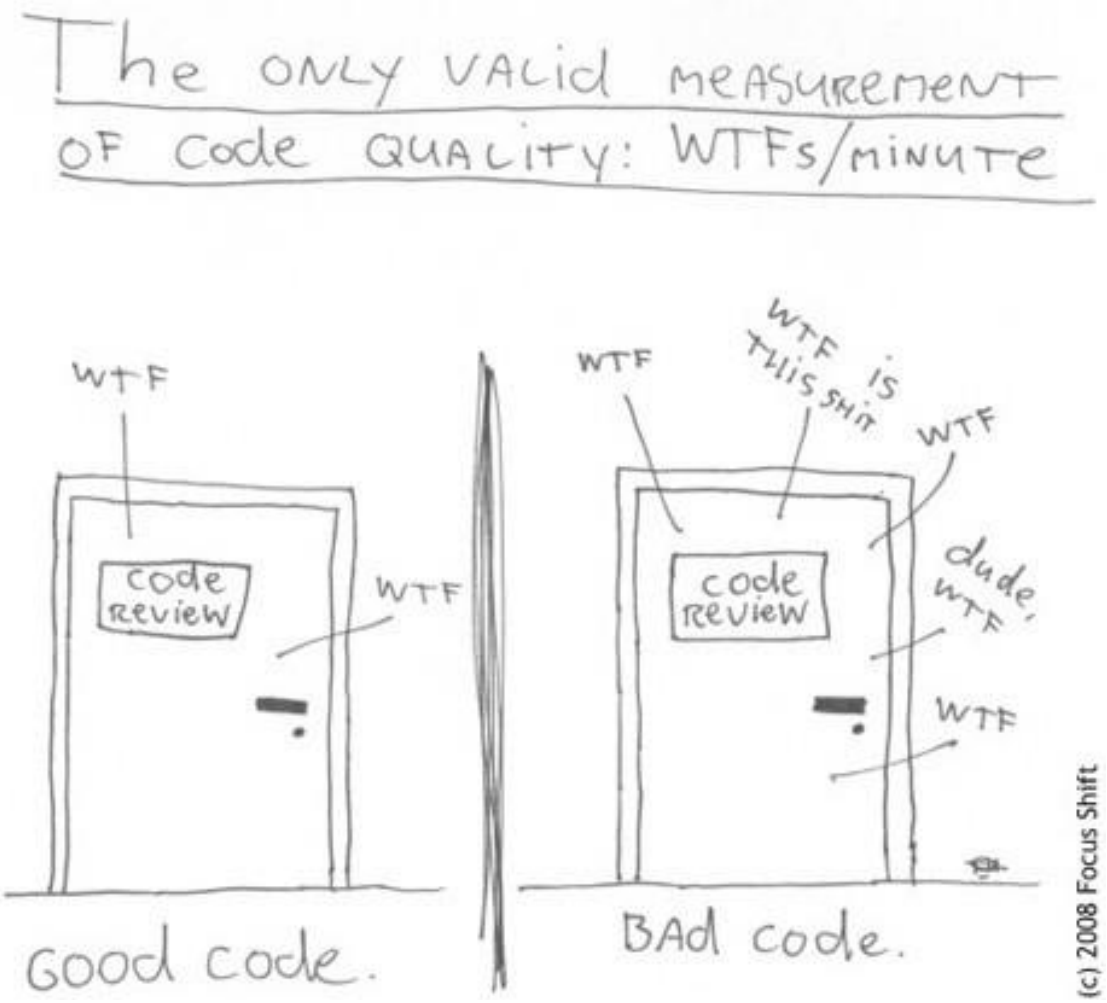


...OR NOT ☺

What happens when code is bad?


TELL ME MORE ABOUT YOUR PROGRAMMER.
Balco
©2008 HTTP://BALOOSCARTOONBLOG.BLOGSPOT.COM

What is a measurement for "good" code?



(c) 2008 Focus Shift

Buzz Group

- What are your experiences of good vs. bad code?
- What do you think are the features?
- Can you measure the quality of code?
- Or is “good coding” more like an art you learn?

Revision from SD1 Syllabus

- **Objects** and **classes**
- **Class definitions**: fields, constructors, methods, parameters
- **Selection** and **iteration**
- **Object interaction**: main method, creating and using class instances.

Declaring classes

```
public class MyClass {
    // fields
    //constructor
    // method declarations
}
```

Example Implementation: Fill in the blanks

```
public class Dog
{
    // Individual characteristics (instance fields).
     ________ String name;

    //constructor
    public Dog(String name)
    {
        this.______=______;
    }
   //accessor method
    private ____ getName(){
        return name;
    }
}
```

FIELDS

CONSTRUCTOR

METHODS

Example Implementation

```
public class Dog
{
     //Individual characteristics (instance fields).
     private String name;

    //constructor
    public Dog(String name)
    {
        this.name=name;
    }
   //accessor method
    private String getName(){
        return name;
    }
}
```

FIELDS

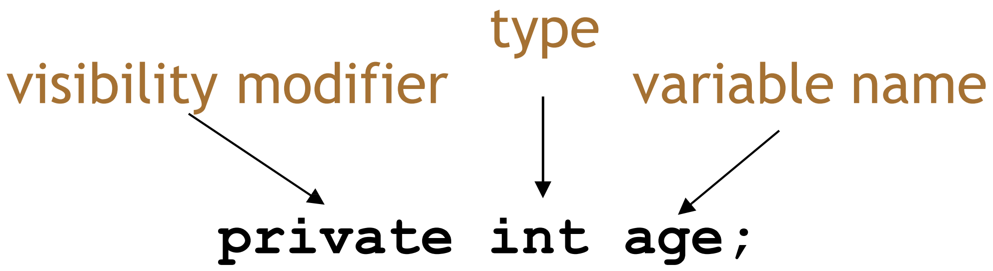
CONSTRUCTOR

METHODS

Fields

```
public class Dog
{
     private String name;
     private int age;

    // Further details omitted.
}
```



type

visibility modifier

variable name

`private int age;`

Constructors

```
public Dog(String name)
{
     this.name = name;
     age = 0;
}
```

- Get called when a new Object is created.
- Store **initial values** into the **fields**.
- Can take external parameter values for this.

Creating new Objects

New Java runtime objects are created by:

1. Using the **new** operator
2. Calling the constructor of a class.
3. Assigning this new instance to a variable of the same type.

actual parameter

<u>Object creation:</u>

1

3

```
Dog fido = new Dog(”Fido”);
```

<u>Associated constructor:</u>

2

```
public Dog(String name);
```

formal parameter

Objects creating objects

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    private String displayString;

    public ClockDisplay()
    {
        hours = new NumberDisplay(24);
        minutes = new NumberDisplay(60);
        …
    }
 //Methods omitted
}
```

declaration

instantiation

```
public class NumberDisplay
{   private int limit;
    private int value;
    Constructor and
    methods omitted.
}
```

Variables

- Member variables in a class—these are called **fields**. Accessible from within one class.

  ```
  private int age = 0;
  ```

- Variables in method declarations—these are called **parameters**. Short lived, local scope.

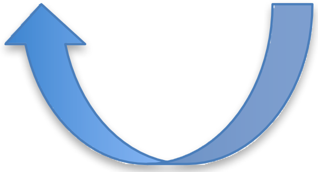
  ```
  private void hunt(Object o){…}
  ```

- Variables within a method or block of code—these `are called` **`local variables.`** Short lived, local scope.

  ```
  int attempt= 0;
  ```

Assignment

- Values are stored into fields (and other variables) via assignment statements:
  - *variable* = *expression*;
  - **`price = cost;`**



Assignments: What is the value?

```
int a = 3;
int b = 4;

b = a;
System.out.println(“B:” + b);

a = b;
System.out.println(“A:” + a);

a +=b;
System.out.println(“A:” + a);
```

Assignments: What is the value?

```
int a = 3;
int b = 0;

b = a;
System.out.println(“B:” + b);
```

B: 3

```
a = b;
System.out.println(“A:” + a);
```

A: 3

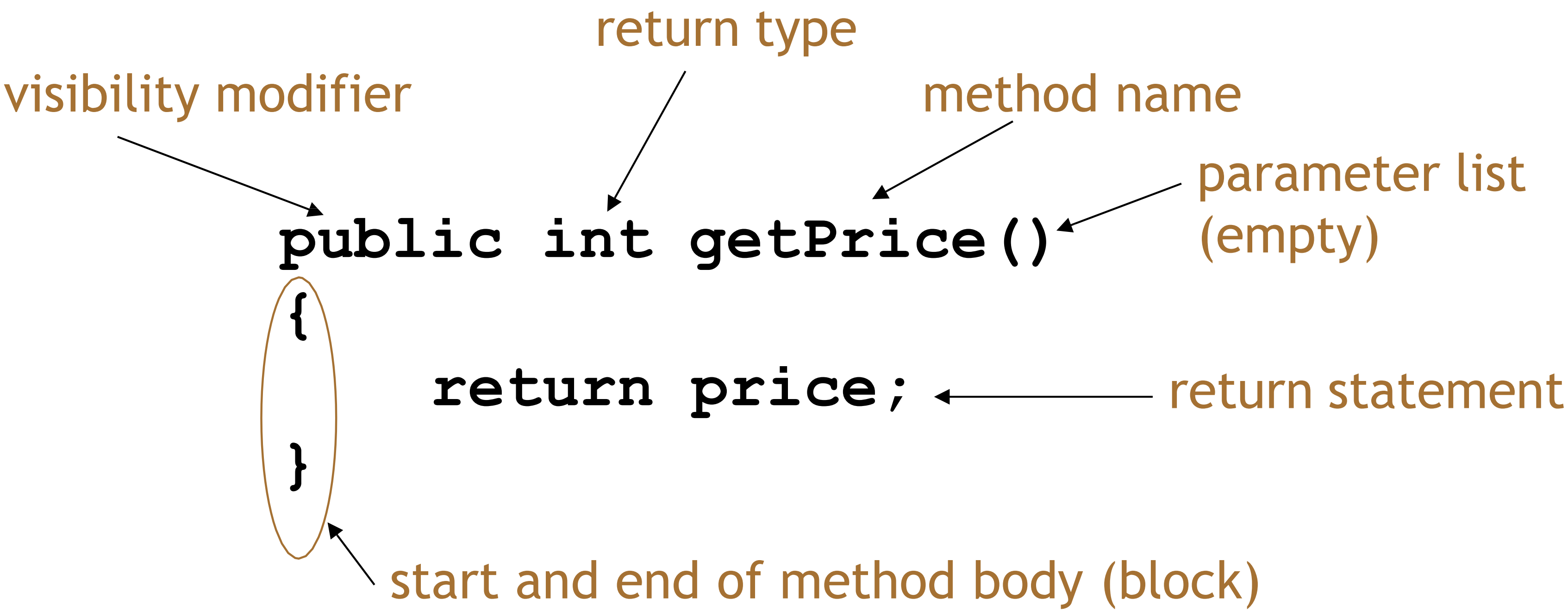
```
a +=b;
System.out.println(“A:” + a);
```

A: 6

Declaring classes

```
public class MyClass {
    // fields ✓
    //constructor ✓
    // method declarations ←
}
```

Accessor (get) methods



return type

visibility modifier

method name

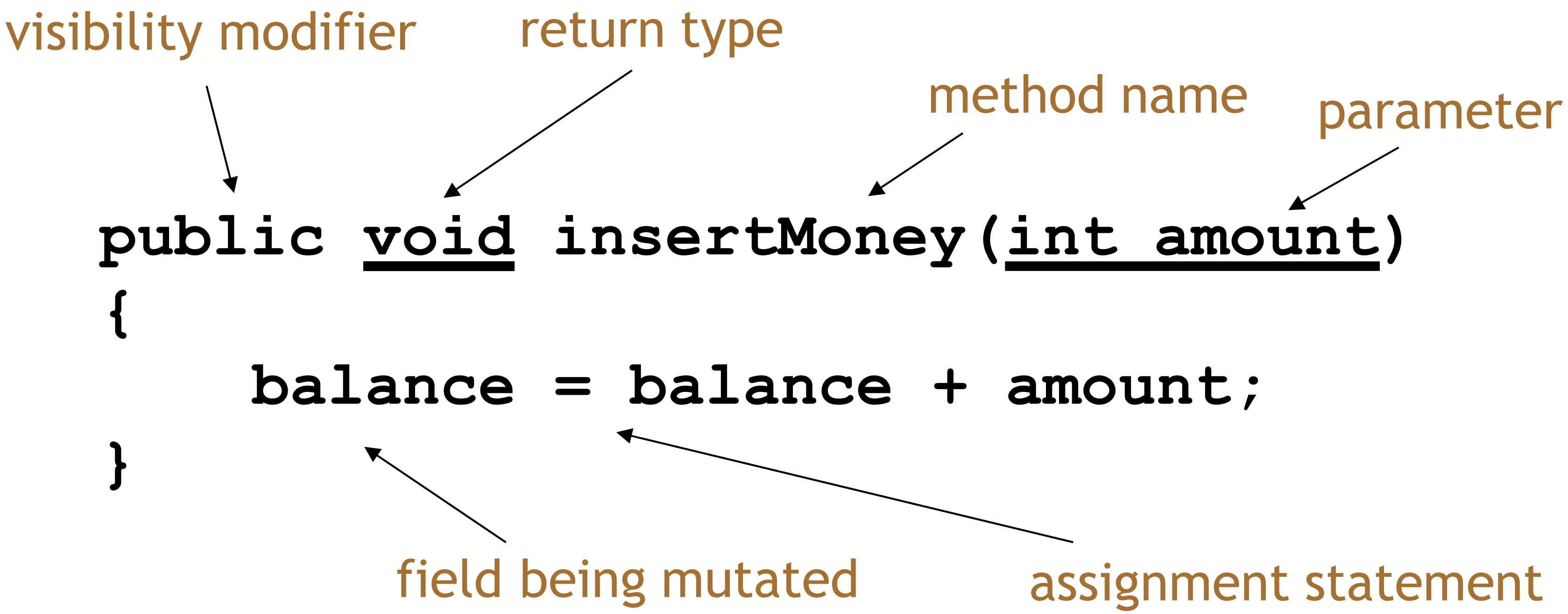
parameter list (empty)

```
public int getPrice()
{
    return price;
}
```

return statement

start and end of method body (block)

Mutator methods



visibility modifier

return type

method name

parameter

```
public void insertMoney(int amount)
{
    balance = balance + amount;
}
```

field being mutated

assignment statement

Homework

- Complete the **self-test** on VISION.

?

GALILEO GALILEI