Understanding Class Definitions

Software Development 1 (F27SA1)

Week 7, lecture 2

*Multiple slides over the course adapted form Verena Rieser @HWU

Last lecture:

- fields
- constructors
- assignments

→ **Let's create a class together!**

Declaring classes

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

Main concepts to be covered this lecture:

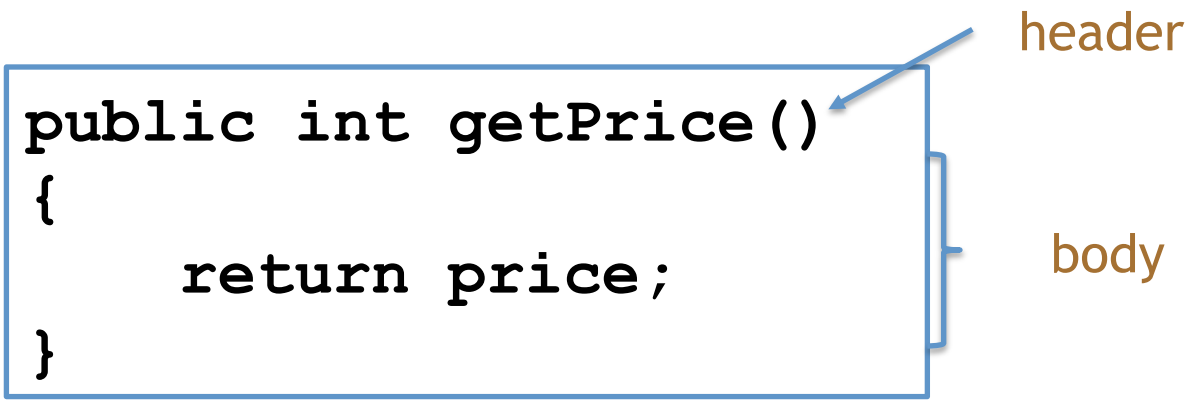
- parameters
- methods
  - including accessor and mutator methods
- ~~scope~~
- ~~local variables~~
- ~~Printing and String concatenation~~

Methods and parameters

- Objects have operations which can be invoked (Java calls them *methods*).
- Methods may have **parameter**s to *pass additional information* needed to execute.

Methods

- Methods implement the behavior of objects.
- Methods have a consistent structure comprised of a *header* and a *body*.



- *Accessor methods* provide information about an object.
- *Mutator methods* alter the state of an object.
- Other sorts of methods accomplish a variety of tasks.

Excursus: Information Hiding

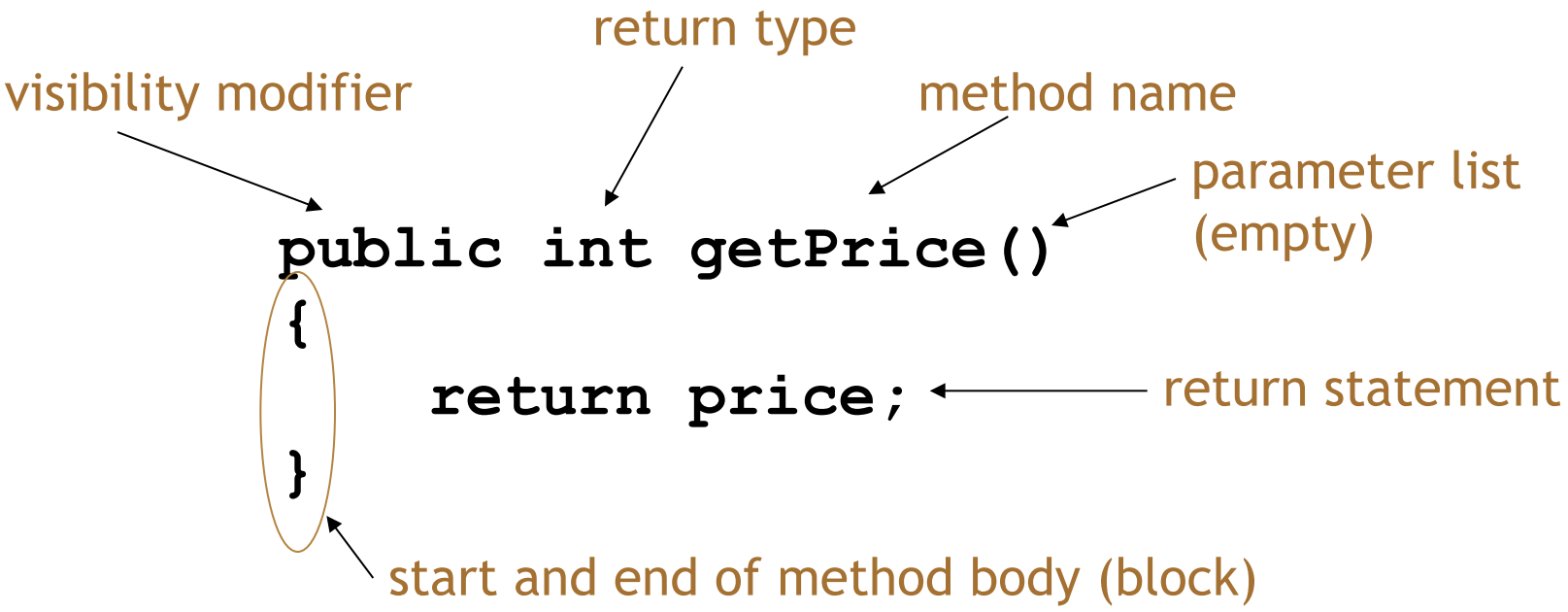
- Information Hiding is one of the Object Oriented Programming principles.
- Other names: *Encapsulation*.
- Why? Makes code more modular and re-usable.
- We will learn about “good design” when programming next year! (SD2)

Method structure

- The header provides the method's ***signature***:
  - `public int getPrice()`

- The header tells us:
  - the name of the method
  - what parameters it takes
  - whether it returns a result
  - its visibility to objects of other classes
- The body encloses the method's statements.

Accessor (get) methods



Accessor methods

- An accessor method always has a **return type** that is <u>not</u> **`void`**.
- An accessor method **returns** a value (*result*) of the type given in the header.
- The method will contain a **`return`** statement to return the value.
- NB: Returning is *not* printing!

Quiz

```
public class CokeMachine
{
    //fields
    private price;

    // constructor
    public CokeMachine()
    {
        price = 300
    }

    // accessor method
    public int getPrice
    {
        return Price;
    }
```

- What is wrong here?

(there are <u>five</u> errors!)

Quiz

```
public class CokeMachine
{
    private price;

    public CokeMachine()
    {
        price = 300;
    }

    public int getPrice()
    {
        return Price;
    }

}
```

- What is wrong here?

(there are five errors!)

Mutator methods

- Have a similar method structure: header and body.
- Used to *mutate* (i.e., change) an object's state.
- Achieved through changing the value of one or more fields.
  - Typically contain assignment statements.
  - Often receive parameters.

Mutator methods

visibility modifier

return type

method name

parameter

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

field being mutated

assignment statement

`set` mutator methods

- Fields often have dedicated **`set`** mutator methods.
- These have a simple, distinctive form:
  - **`void`** return type
  - method name related to the field name
  - single parameter, with the same type as the type of the field
  - a single assignment statement

A typical `set` method

```
public void setDiscount(int amount)
{
    discount = amount;
}
```

Q: What can we **infer** about the variable `discount` ?

We can infer that `discount` is a <u>field</u> of <u>type `int`</u>, i.e:

```
private int discount;
```

Protective mutators

- A set method does not have to assign the parameter to the field.
- The parameter may be checked for validity and rejected if inappropriate.
- Mutators thereby protect fields.
- Mutators support *encapsulation*.

Example: Protection

```
public void setDiscount(int amount)
{
    if(amount < price){
        discount = amount;
    }
    else{
        System.out.println("You're discounting too much!")
    }
}
```

Method summary

- Methods implement all object behaviour.
- A method has a name and a return type.
  - The return-type may be **`void`**.
  - A non-**`void`** return type means the method will return a value to its caller.
- A method might take parameters.
  - Parameters bring values in from outside for the method to use.

Homework

- Read chapters:
  - 2.7-2.23
  - 3.8.2 on “String concatenation”
  - You can leave out: 2.15 and 2.22. we will talk about this later