Agenda

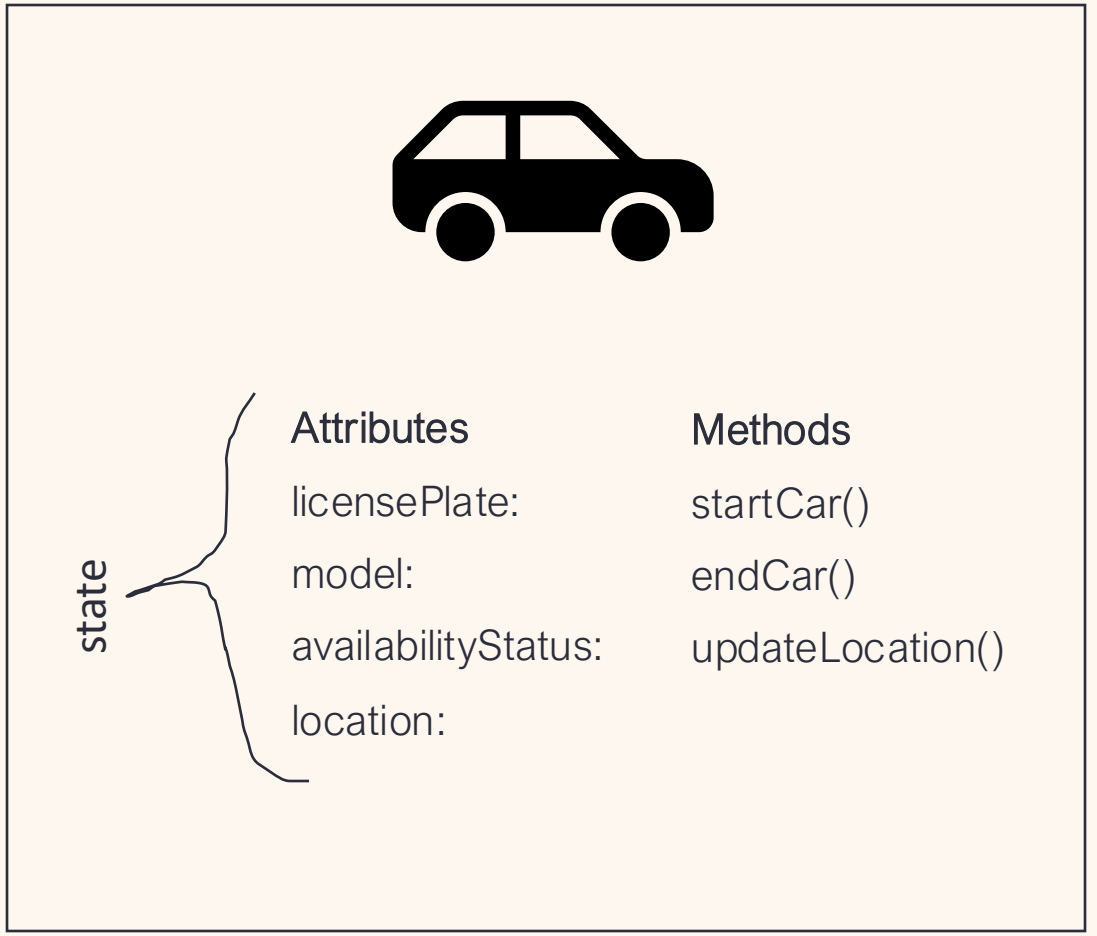
- Recap
- Variables and instance variables
- Declaration, Intialisation
- Constructor, methods, parameters
- ...

Objects / Instances

Each object has a set of methods (implement the behaviour of an object).

An object has a state that consists of a set of data values.

Method calls often result in a change of state of an object.



Example: Car sharing system

1. User
   - Attributes:
   - Methods:
2. Car
   - Attributes:
   - Methods:
3. Trip
   - Attributes:
   - Methods:
4. Payment
   - Attributes:
   - Methods:

Example: Car sharing system

1. User
   - Attributes: name, membershipType, currentLocation
   - Methods: reserveCar(), startTrip(), endTrip(), payForTrip()

2. Car
   - Attributes: carID, licensePlate, model, availabilityStatus, location
   - Methods: isAvailable(), startCar(), endCar(), updateLocation()

3. Trip
   - Attributes: user, car, startTime, endTime, distance
   - Methods: calculateFare(), startTrip(), endTrip()

4. Payment
   - Attributes: user, trip, amount
   - Methods: processPayment(), generateReceipt()

Review

Classes model concepts.Source code implements those concepts.

Source code defines:

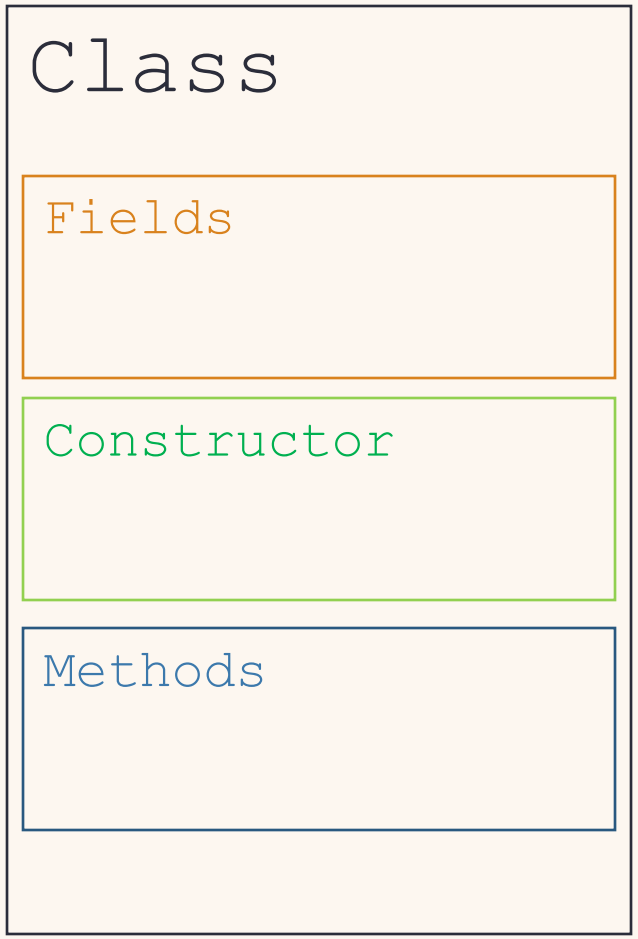
- What objects can do (methods).
- What data they store (attributes).

Objects come into existence with pre-defined attribute values.

The methods determine what objects do with their data.

Elements of a Class

```
public class ClassName
{
    Fields

    Constructors

    Methods



}
```



Basic class structure

```
public class ClassName
{
    Fields
    Constructors
    Methods
}
```

The outer wrapper of a class

The inner contents of a class

Objects / Instances

Each object has a set of methods (implement the behaviour of an object).

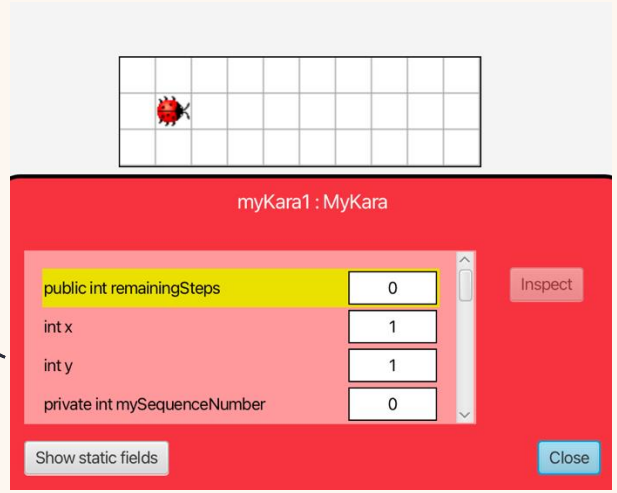
An object has a state that consists of a set of data values.

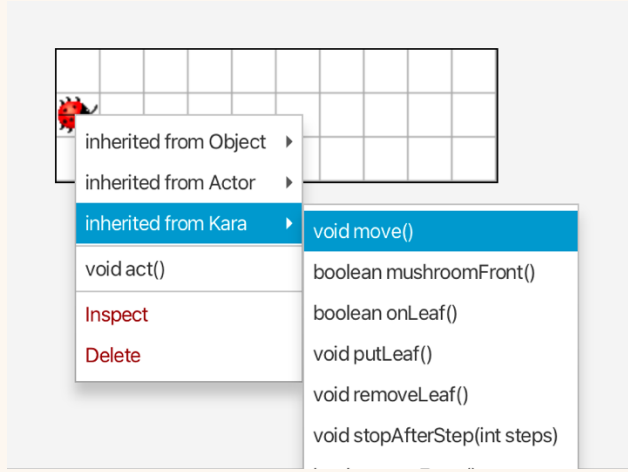
Method calls often result in a change of state of an object.

Attributes

Methods

state





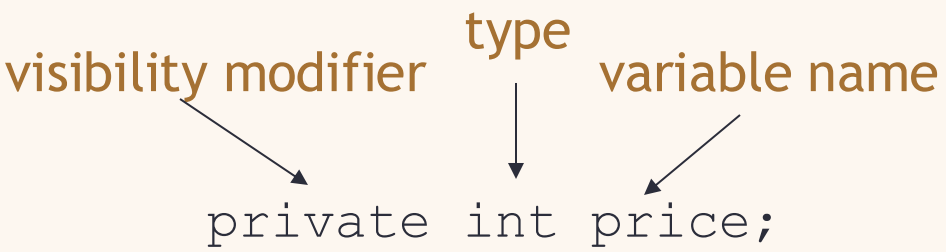
Variables

- A variable is a storage location.
- Variables store values that can change during the runtime of the program.
- In general, variables consist of three components:
  - Identifier (name of the variable)
  - Data type
  - Value (current content of the variable, “literale”)

```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    …
}
```

Instance variables (fields)

- Instance variables store values for an object.
- Define the state of an object.
- Some values change often.
- Some change rarely (or not at all).



Choosing variable names

There is a lot of freedom over choice of names. Use it wisely!

Choose expressive names to make code easier to understand:

- **`price`**, **`amount`**, **`name`**, **`age`**, etc.

Avoid single-letter or cryptic names:

- **`w`**, **`t5`**, **`xyz123`**

In compound words, new word parts should begin with capital letters (CamelCase), e.g., “valueEntered.”

`Java keywords (new, private, int…) cannot be used!`

Declaration and Initialization

The statement: `int price = 3;`

- creates a variable of type **int**
- named **price** (declaration)
- assigns it the initial value of **3** (initialization).

Declaration of variables without initialization is possible:
`private int price;`

Only primitive data types have a default value!

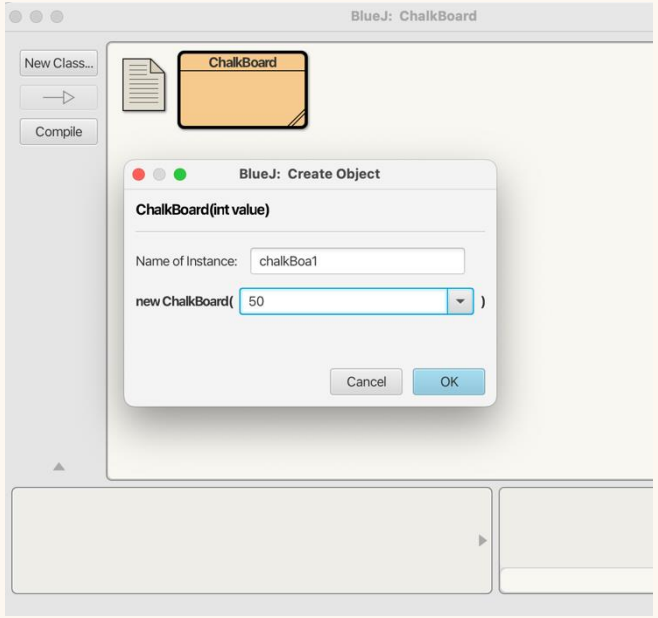
Constructors

- Initialize an object.
- Have the same name as their class.
- Close association with the fields:
  - Initial values stored into the fields.
  - Parameter values often used for these.

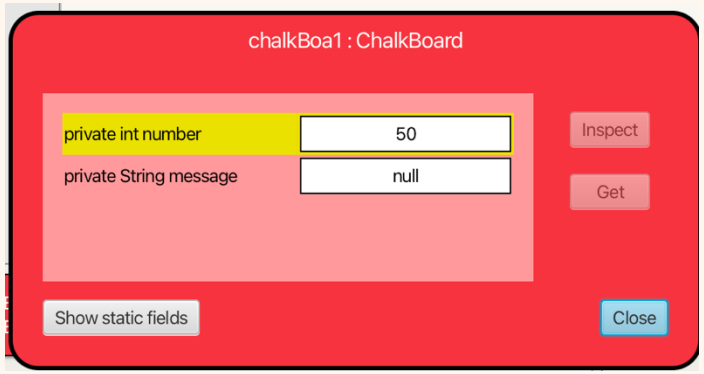
```
public ChalkBoard(){

      number = 60;
      message = ”hallo";
}
```

```
public ChalkBoard(int value)
{
    number = value;
}
```

Passing data via parameters




```
public ChalkBoard(int value)
{
    // initialise instance variables
    number= value;
}
```



Parameters are another sort of variable.

Assignment

Values are stored into fields (and other variables) via assignment statements:

- *variable = expression;*

- **`number = value + 1;`**

A variable can store just one value, so any previous value is lost.

Zustand einer Instanz (eines Objektes) ändern

????

Method structure

The header:

- **`public void setNumber( int value)`**

The header tells us:

- the *visibility* to objects of other classes;
- whether the method *returns a result*;
- the *name* of the method;
- whether the method takes *parameters*.

The body encloses the method's *statements*.

- {}

`set` mutator methods

Fields (instance variables) often have dedicated **`set`** (mutator) methods.

These have a simple, distinctive form:

- **`void`** return type
- method name related to the field name
- single formal parameter, with the same type as the type of the field
- a single assignment statement

```
public void setNumber(int value)
{
    number = value;
}
```

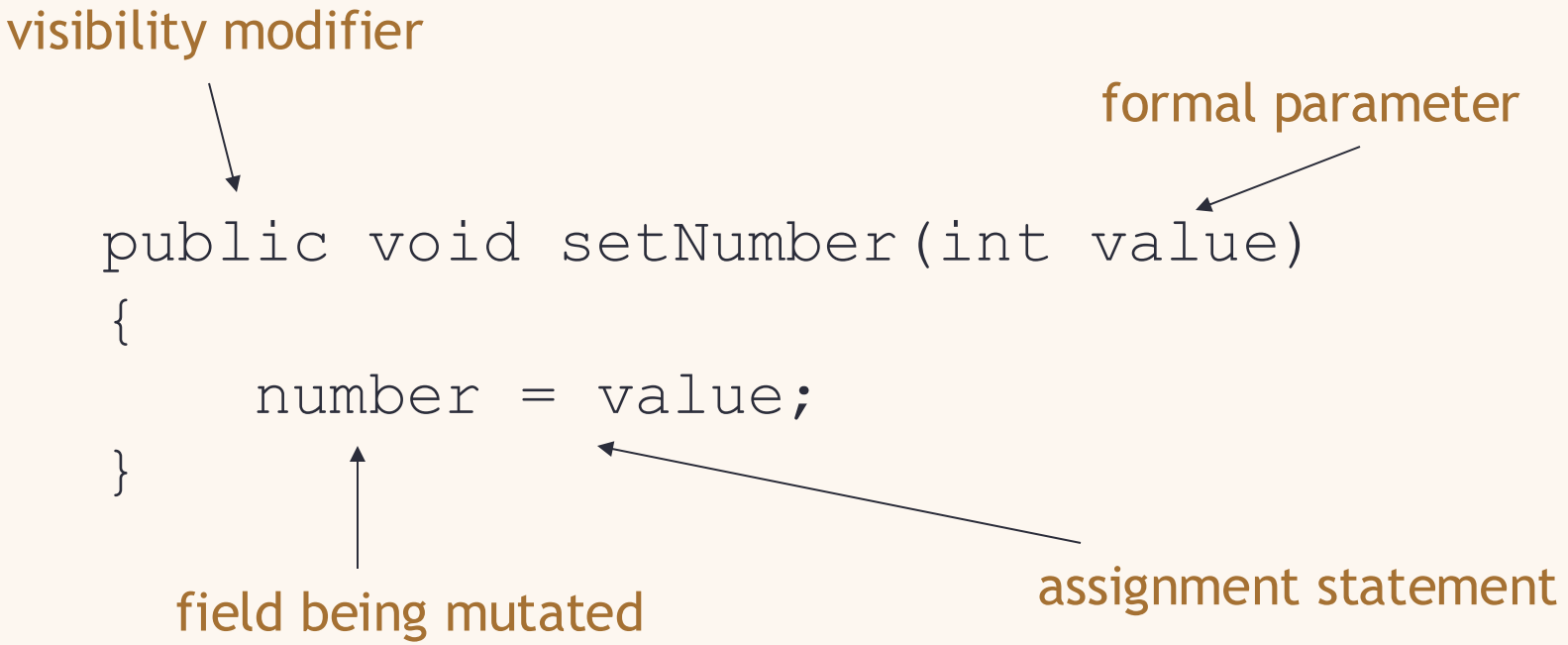
Mutator methods (setter)

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.

- They typically contain one or more assignment statements.
- Often receive parameters.



A typical `set` method

```
public void setNumber(int value)
{
    number = value;
}
```

We can easily infer that `discount` is a field of type `int`, i.e:

```
private int discount;
```

Try out!

Write a method for your chalkboard class that changes an attribute.

Methods

Methods implement the *behavior* of objects.

Methods have a consistent structure comprised of a *header* and a *body*.

Mutator methods (setter) alter the state of an object.

Accessor methods (getter) provide information about an object.

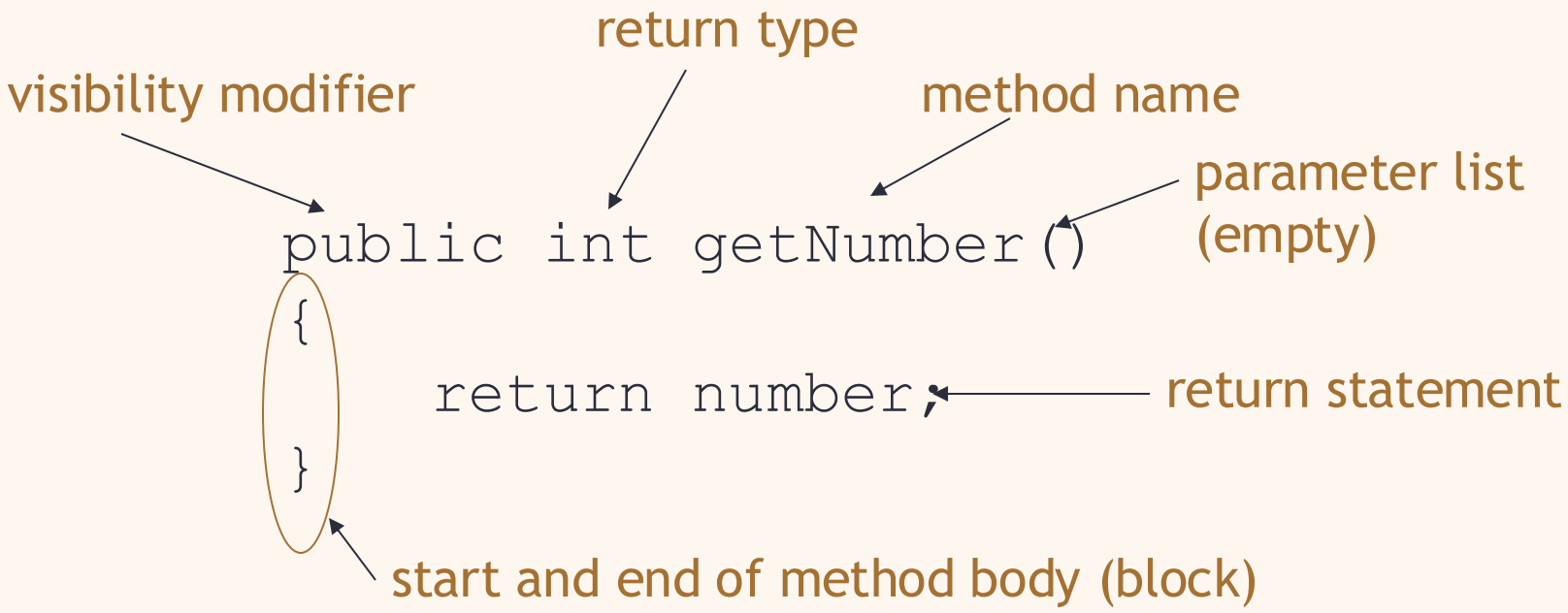
Other sorts of methods accomplish a variety of tasks.

Zustand einer Instanz (eines Objektes) abfragen

????

Accessor (**`get`**) methods

- An accessor method always has a return type that is not **`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!



Accessor methods

An accessor method always has a return type that is not **`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!

Test

What is wrong here?

```
public class CokeMachine
{
private price;

public CokeMachine()
{
   price = 300
}

public int getPrice
{
   return Price;
}
```

(there are five errors!)

Test

```
public class CokeMachine
{
   int
private price;

public CokeMachine()
{
   price = 300;
}

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

- What is wrong here?

(there are five errors!)

Method summary

Methods implement all object behavior.

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.

Agenda

- Recap: getter / setter Methods
- Ticket machine project: review
- Conditions: if else
- Ticket machine project: improve with conditions
- …

Previously on Info I

Source code

Each class has source code associated with it that defines its details

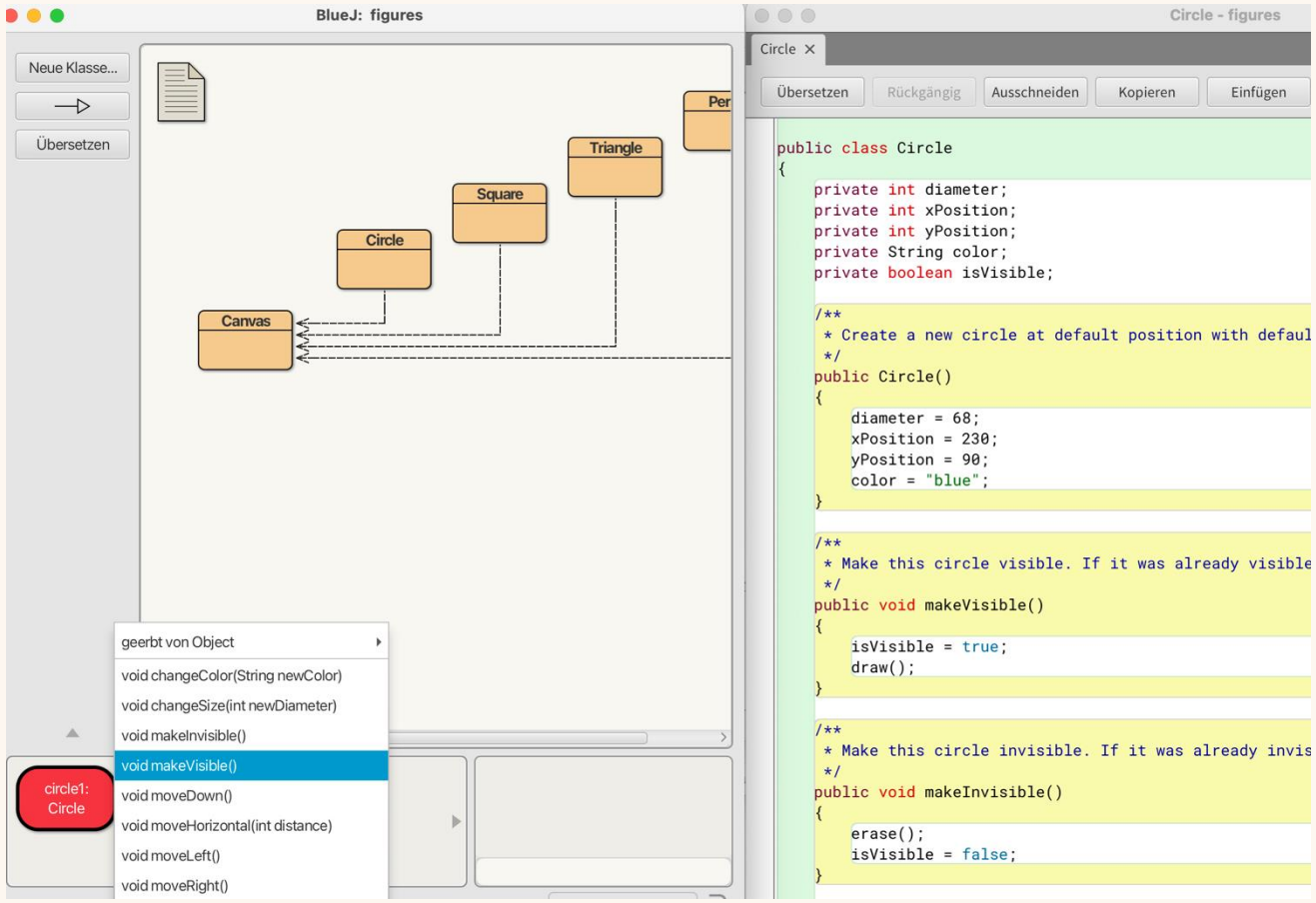
- _____ and
- _____

How can we change the state of an instance?

Previously on Info I

Source code

How can we change the state of an instance?

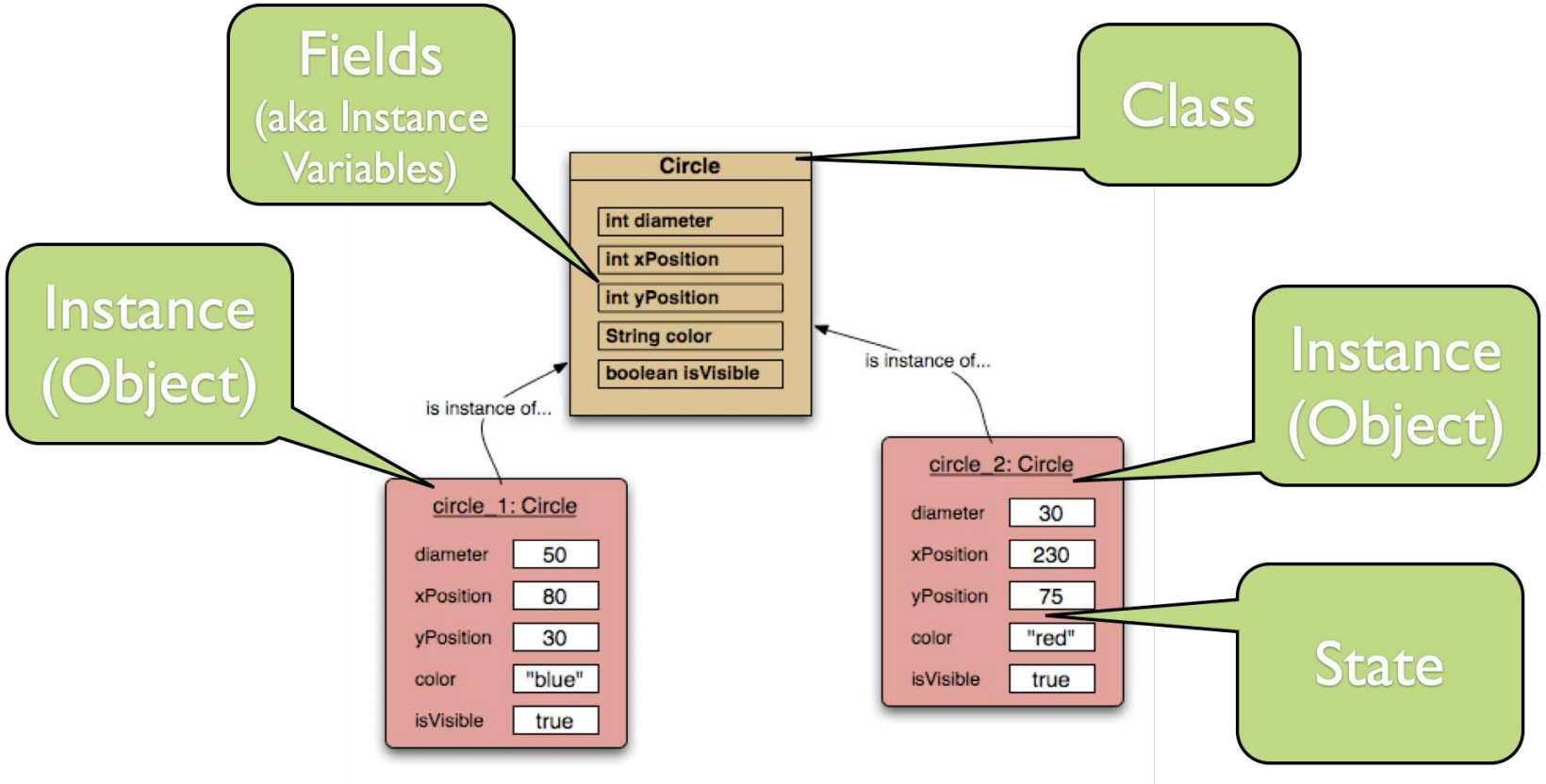


Review

Some methods take _____ that affect their actions.

Some methods return a _____.

Class definition and instance variables



Setter and getter methods

Field / Instance variable

```
private int number;
```

setter method, to change value

method name

parameter list

```
public void setNumber(int num)
{
    number = num;
}
```

assignment

getter method, to get value

return type

method name

```
public int getNumber()
{
    return number;
}
```

return statement

Ticket machines - modeling

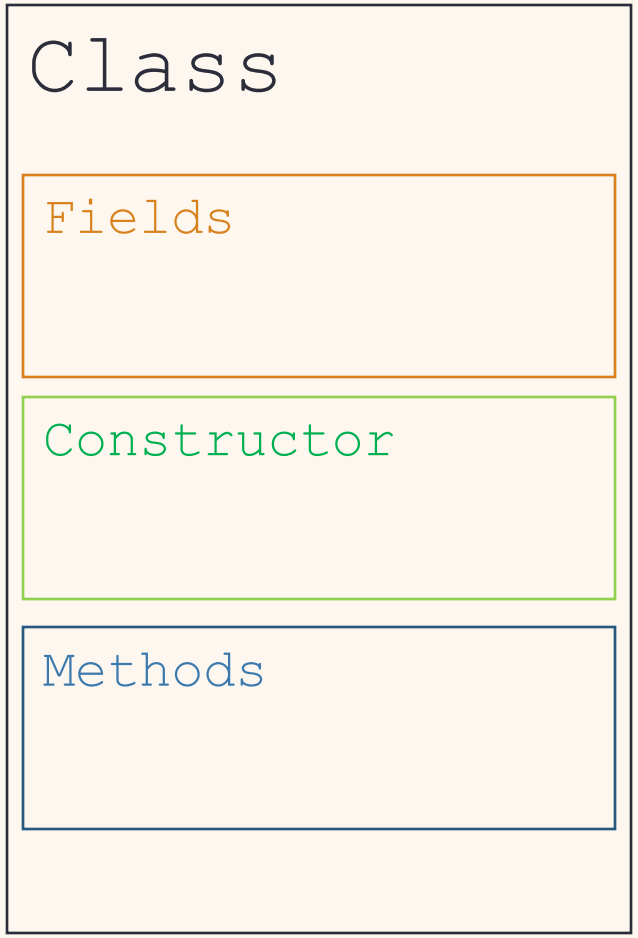
Welche Attribute benötigt ein einfacher Fahrkartenautomat?

Welche Methoden benötigt der Automat?

Murmelphase…

Ticket machines – an internal view

- Interacting with an object gives us clues about its behavior.
- Looking inside allows us to determine how that behavior is provided or implemented.
- All Java classes have a similar-looking internal view.



Ticket machines – an external view

Exploring the behavior of a typical ticket machine.

- Use the *naive-ticket-machine* project.
- Machines supply tickets of a fixed price.
  - How is that price determined?
- How is ‘money’ entered into a machine?
- How does a machine keep track of the money that is entered?

Ticket machines

Demo of naïve-ticket-machine

Basic class structure

```
public class TicketMachine
{
    Inner part omitted.
}
```

The outer wrapper of TicketMachine

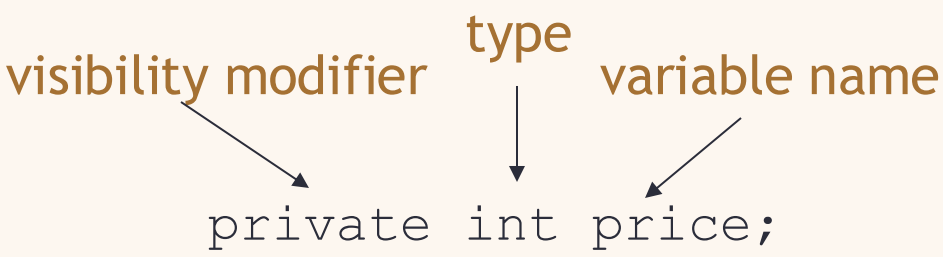
```
public class ClassName
{
    Fields
    Constructors
    Methods
}
```

The inner contents of a class

Instance variables (fields)

- Instance variables store values for an object.
- Define the state of an object.
- Some values change often.
- Some change rarely (or not at all).

```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    Further details omitted.
}
```



Exercise

2-20 Add a showPrice method to the TicketMachine class.

- This should have a void return type and take no parameters. The body of the method should print something like: The price of a ticket is xyz cents.
- Where xyz should be replaced by the value held in the price field when the method is called.

2-21 Create two ticket machines with differently priced tickets.

- Do calls to their showPrice methods show the same output or different? How do you explain this effect?

2-22 What do you think would be printed if you altered the fourth statement of printTicket so that price also has quotes around it, as follows?

System.out.println("# " + "price" + " cents.");

String concatenation (Verkettung) vs arithmetic operator

```
4 + 5
 9
"wind" + "ow"
 "window"
"Result: " + 6
 "Result: 6"
"# " + price + " cents"
 "# 500 cents"
```

→ overloading

- Different ways to overload the method
- By changing the no. of arguments
- By changing the datatype

Quiz

```
System.out.println(5 + 6 + "hello");
```

```
11hello
```

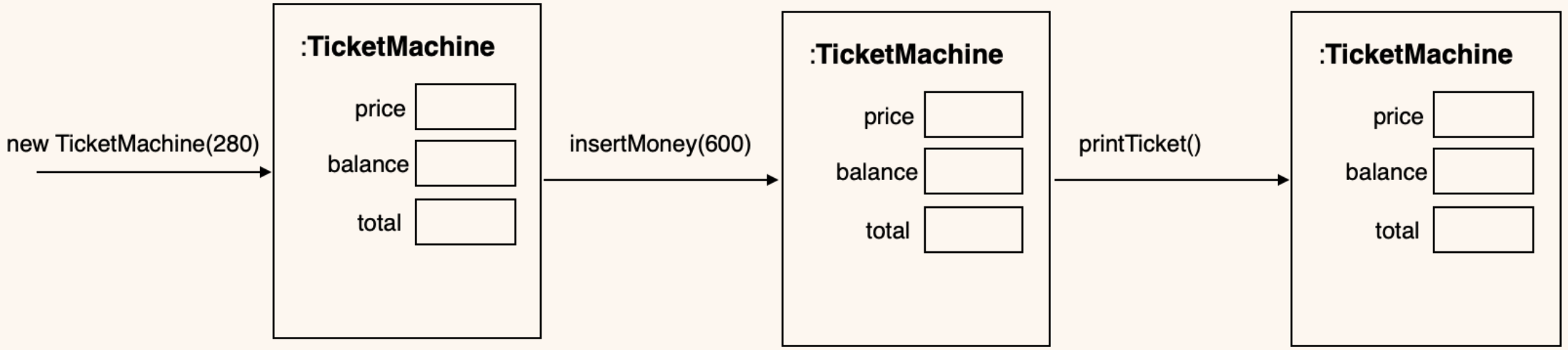
```
System.out.println("hello" + 5 + 6);
```

```
hello56
```

Examine the class TicketMachine

Was funktioniert nicht so wie erwartet?

Original Version: Why is the Total computed incorrectly? Fill in the field values after each method has completed and try figure out where it should be changed to compute the total correctly.



Reflecting on the ticket machines

Their behavior is inadequate in several ways:

- No checks on the amounts entered.
- No refunds.
- No checks for a sensible initialization.

How can we do better?

- We need the ability to choose between different courses of action.

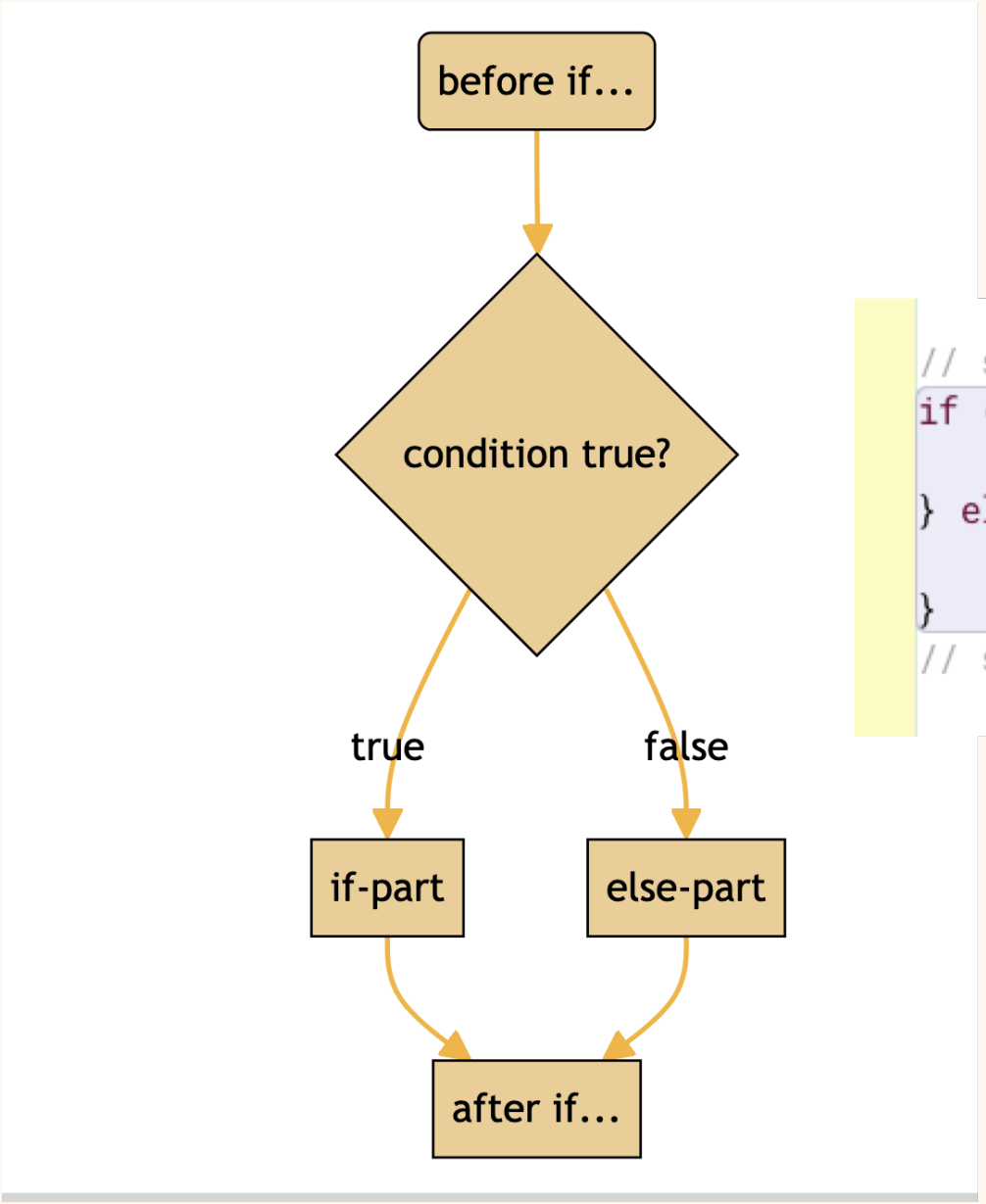
Making choices in everyday life

If I have enough money left, then I will go out for a meal
otherwise I will stay home and watch a movie.

Making a choice in everyday life

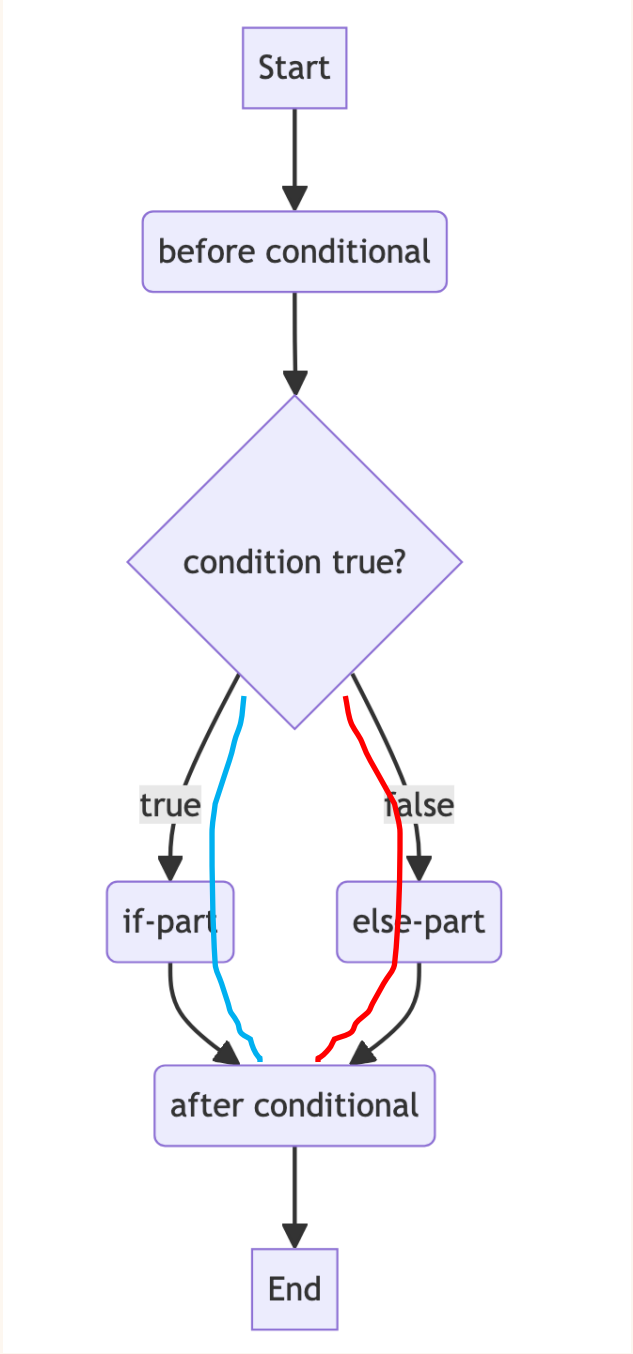
```
if(I have enough money left) {
    I will go out for a meal;
} else {
    I will stay home and watch a movie;
}
```

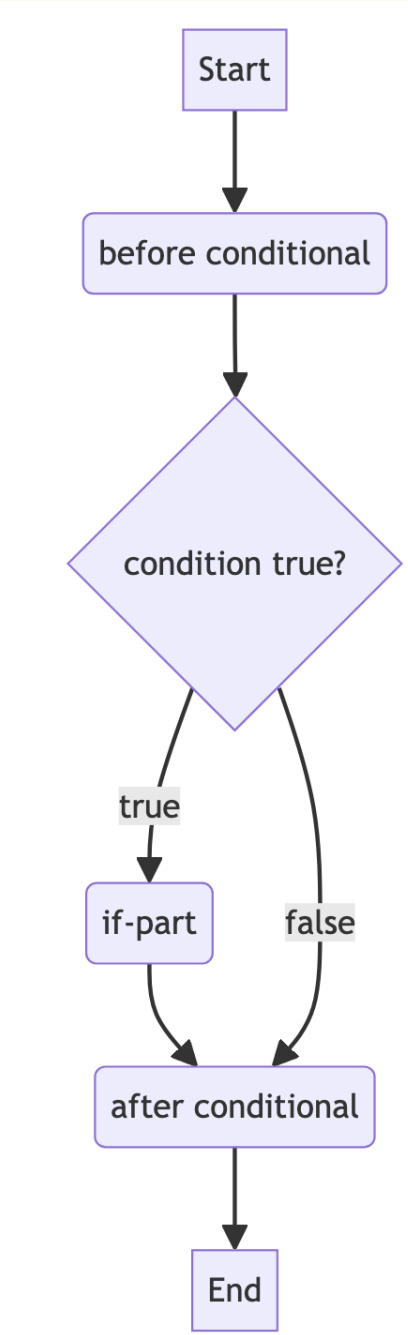
If else



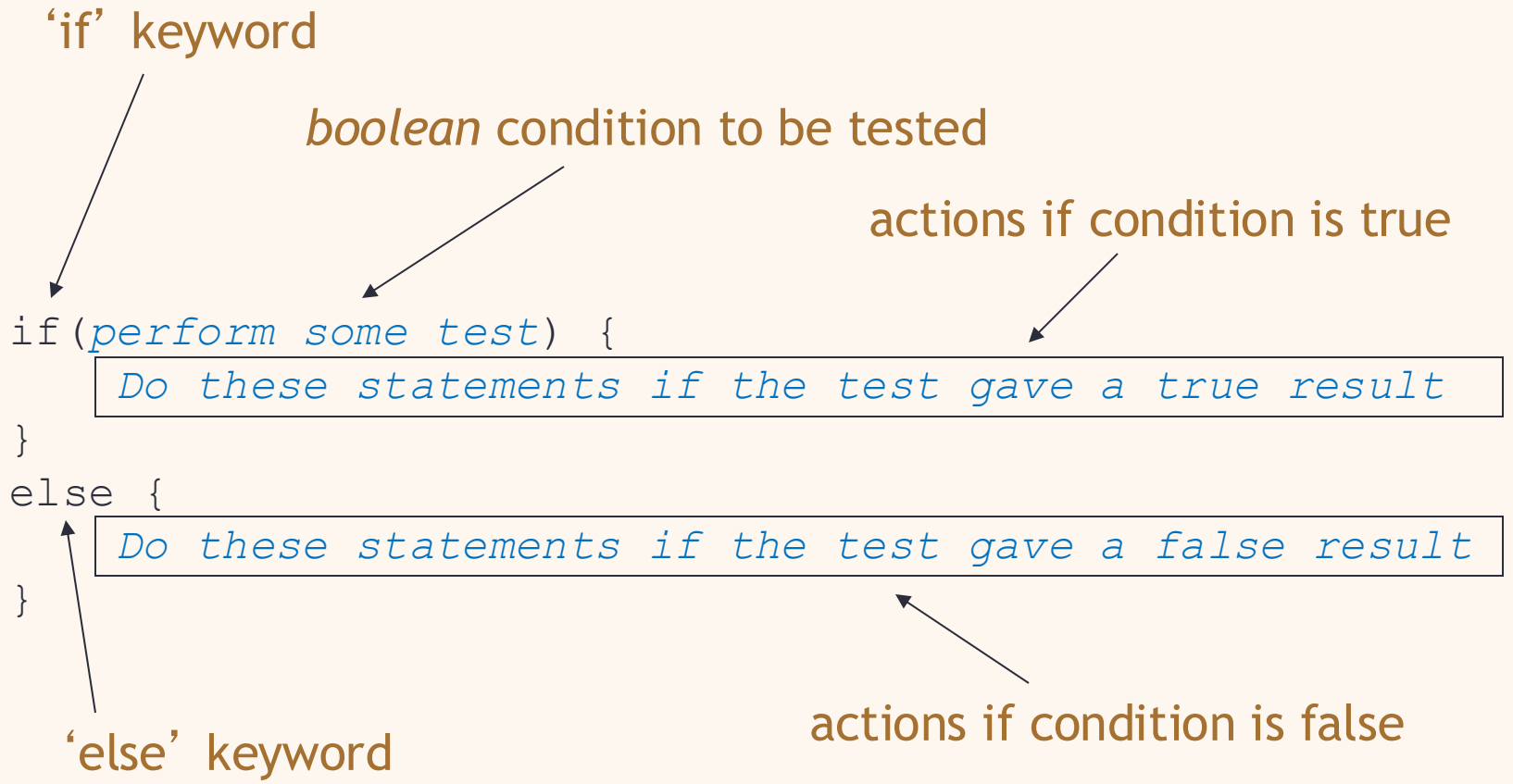

```
// statements before if...
if (condition) {
    // if-part
} else {
    // else part
}
// statements after if....
```

If/else Activity Diagrams





Making choices in Java



Ausdrücke / Operatoren / Vergleiche

Integer-Arithmetik (int, short, long):	`+, -, *, /, %`
Gleitkomma-Arithmetik (float, double):	`+, -, *, /`
Boolesche Arithmetik (boolean):	`&&, ||, !`
Vergleichsoperatoren:	`==, !=, <, >, <=, >=`
Zuweisungsoperatoren:	`=, +=, -=, *=, /=, %=`
Inkrement-Operator:	`++`
Dekrement-Operator:	`--`
Bit-Operatoren:	`<<, >>>, &, |, ~, ^, ...`
Spezielle Operatoren:	`?:, (type)`

Making a choice in the ticket machine

```
public void insertMoney(int amount)
{
    if(amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println("Use a positive amount: " + amount);
    }
}
```

conditional statement avoids an inappropriate action

Exercise: Improve the TicketMachine

First, Identify Constructors, Fields, Mutators and Accessors in the class.

Then find the locations where you can fix the TicketMachine:

1 No checks on the amounts entered.

- 1a check if that the inserted amount is not negative
- 1b only issue a ticket if enough money was inserted

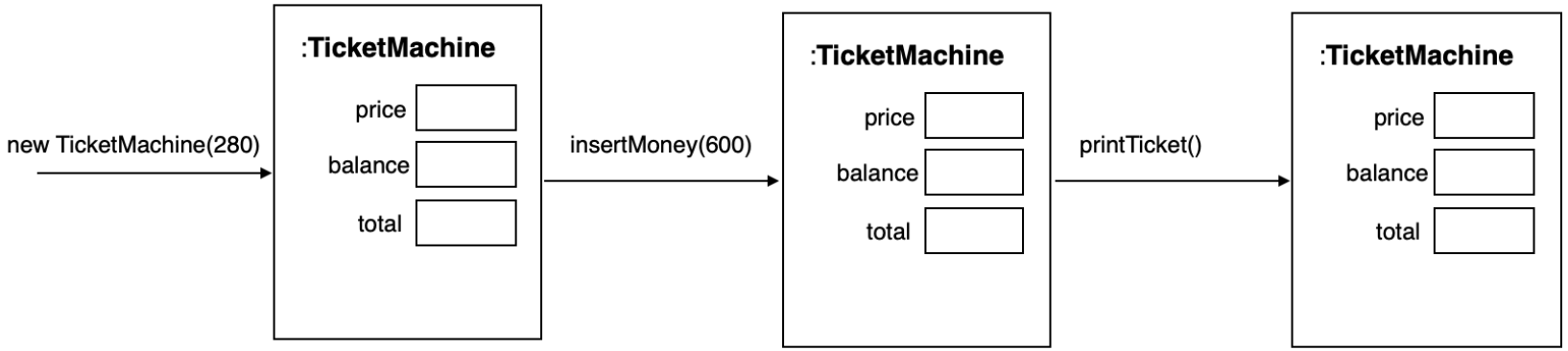
2 No checks for a sensible initialization.

- 2a check for a sensible price given to the constructor

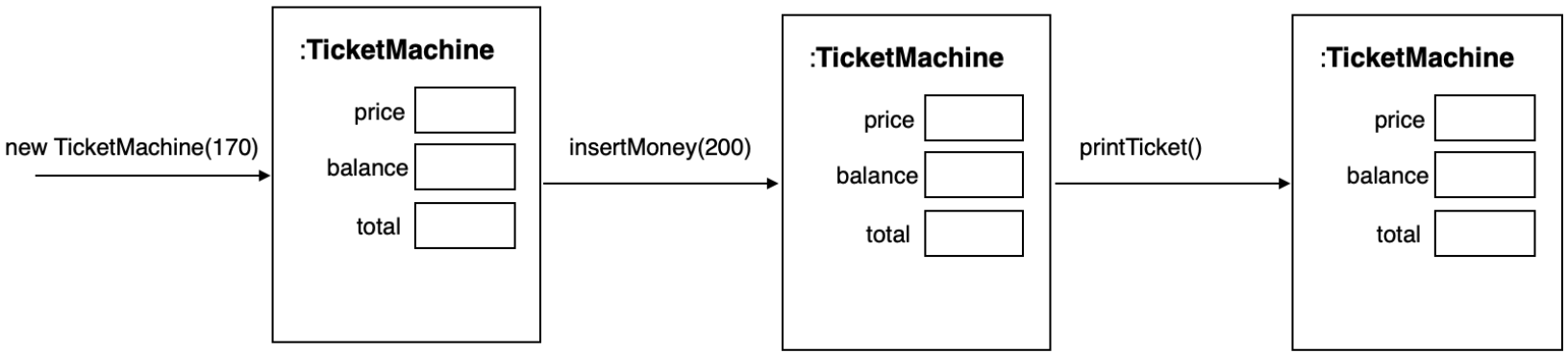
3 No refunds.

- 3a balance should not be set to 0 after ticket is printed.

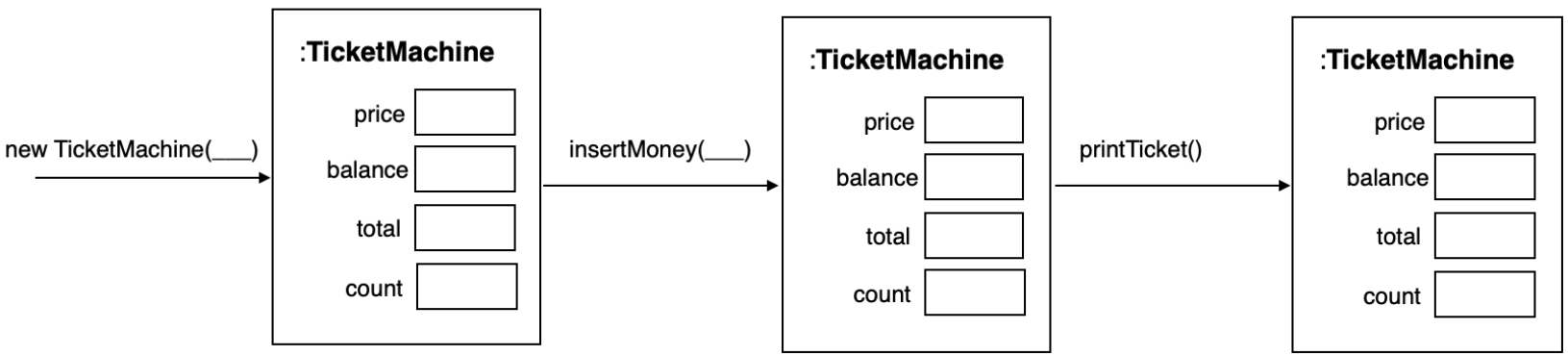
Original Version: Why is the Total computed incorrectly? Fill in the field values after each method has completed and try figure out where it should be changed to compute the total correctly.



Perform a walkthrough with your corrected version.



Add and test code to count the tickets issued.



Starting with BlueJ

Exploring fundamental concepts via BlueJ:

- https://www.youtube.com/watch?v=Q1BuFi4UvpQ

All videos

- https://www.youtube.com/playlist?list=PL8LRe866vedtI5vM5iheAKzItvu9qZyb_