Objects First With Java
A Practical Introduction Using BlueJ

Understanding class definitions

Looking inside classes

Looking inside classes

- basic elements of class definitions
  - fields
  - constructors
  - methods
- methods and constructors: parameters for passing data
- methods contain statements ("action")
  - printing statements
  - conditional statements
  - assignment statements

Main concepts to be covered

- fields/attributes
- constructors
- methods
- parameters
- assignment statements
- conditional statements

Example: project “ticket machine”

- one class only: TicketMachine
- What’s it all about?
  - machine sells one type of tickets
  - one price
  - collects the correct amount of money.

Ticket machines – an external view

- Exploring the behaviour of a typical ticket machine.
  - 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 has been entered so far?

Ticket machines – an internal view

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

Basic class structure

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

The outer wrapper of TicketMachine

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

The contents of a class

(order of appearance is natural, but not mandatory)

Fields,constructors, & methods

- Fields characterize an object, i.e. contain specific values.
- Constructors initialize objects by assigning initial values to their fields.
- Methods implement the behaviour of objects ➔ any kind of further action after the initialization.

Fields

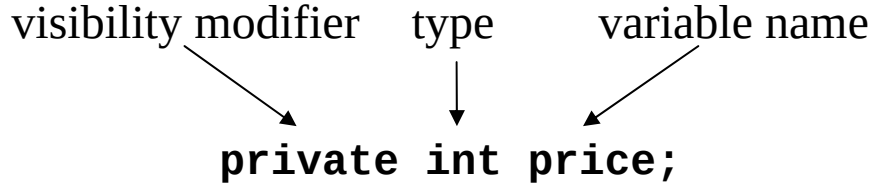
- Fields store values for an object.
- Fields are also known as instance variables ("variable": values can change!).
- Use the *Inspect* option to view an object's fields.
- Fields define the state of an object.
- Visibility defines scope

meaning?

semantics?

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

    Constructor and methods omitted.
}
```



Attention: Comments!

- Comments are important for code documentation.
- Adding comments is a question of good programming style.
- One-line comments: `//` ...
- Multi-line comments: `/*` ... `*/`
- (javadoc comments: `/**` ... `*/`)

Constructors

- ... initialize an object.
- ... have the same name as their class.
- ... store/assign initial values into the fields.
- ... often receive external parameter values

  (here: `ticketCost`)

```
public TicketMachine(int ticketCost)
{
    price = ticketCost;
    balance = 0;
    total = 0;
}
```

external parameters:
transfer of information
from outside a class to
inside!
`ticketCost`: formal parameter
500: actual parameter

Constructors

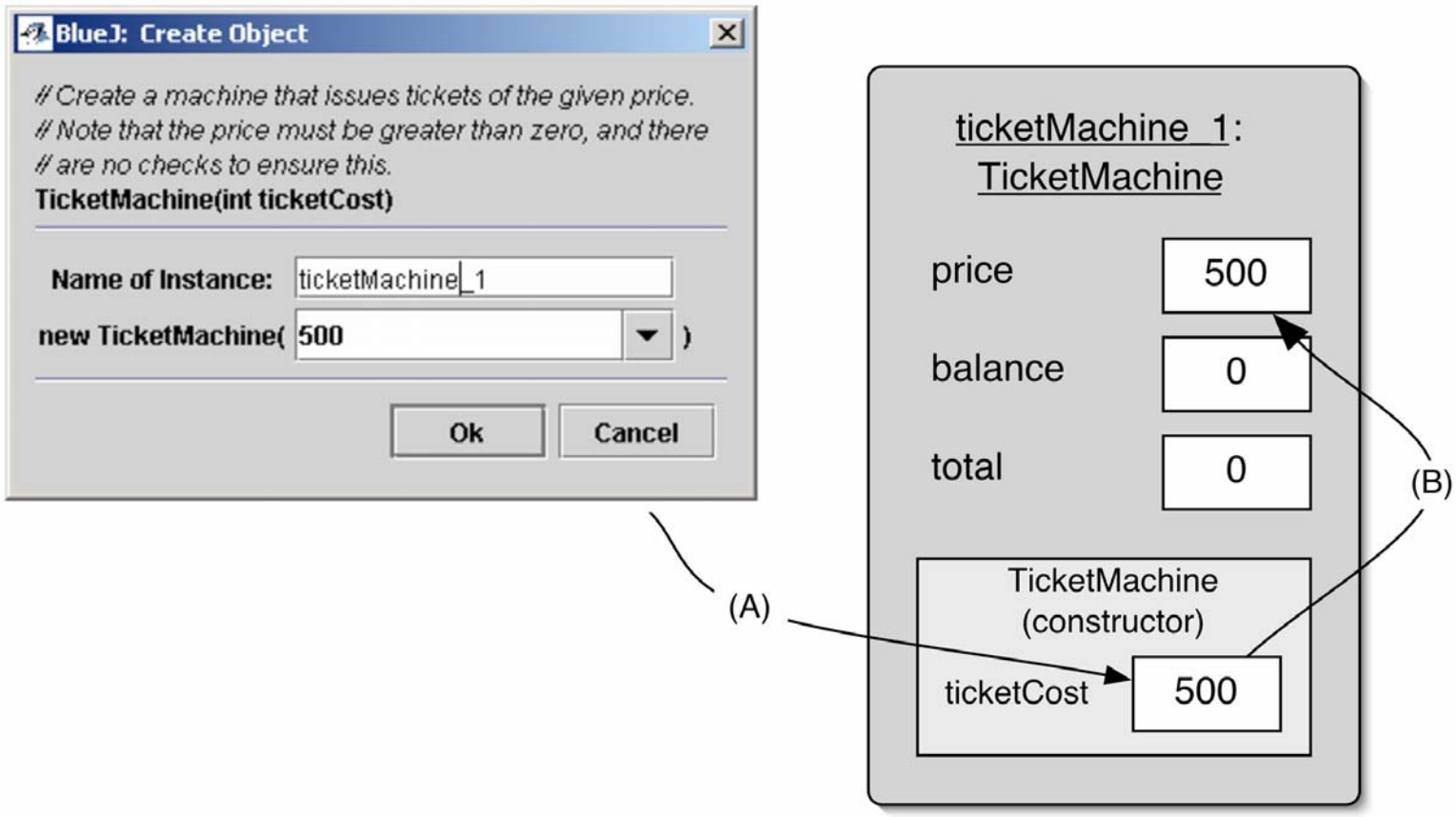
- ... initialize an object.
- ... have the same name as their class.
- ... store/assign initial values into the fields.
- ... often receive external parameter values

  (here: `ticketCost`)

```
public TicketMachine(int ticketCost)
{
    price = ticketCost;
    balance = 0;
    total = 0;
}

public class Main
{
    public static void main()
    {
        TicketMachine myMachine =
            new TicketMachine(500);
        /* …*/
    }
}
```

Passing data via parameters



Assignment

- a prominent example of a statement
- Values are stored into fields (and other variables) via assignment statements:
  - *variable* = *expression;*
  - `price = ticketCost;`
- A variable stores a single value ➔ any previous value is lost!
- Assignment operators: `=, +=, -=`

Data types

- So far, we've seen `int`, `string`, and `boolean` (do you know more?).
- The type of the expression on the right-hand side must match the variable's type!
- otherwise: `true = 17.84` ???

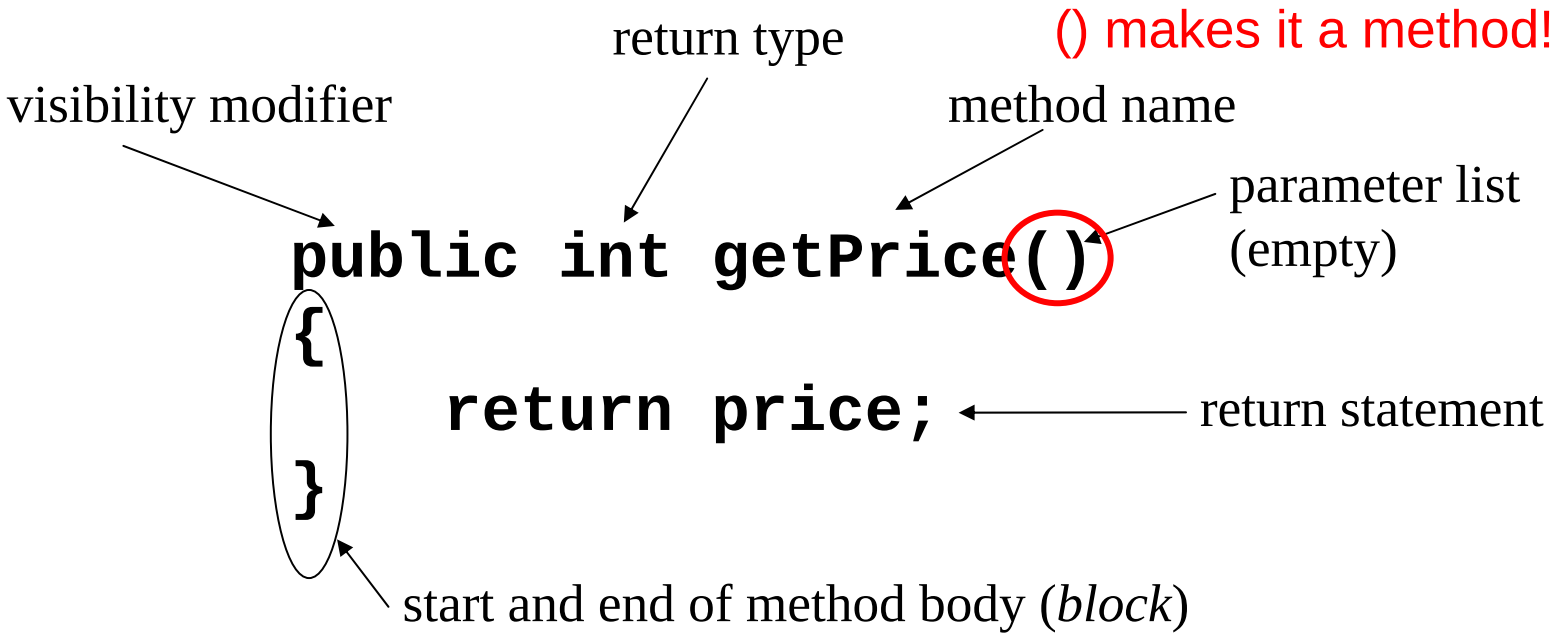
Methods

- Our `TicketMachine` has four methods:
  - `getPrice`
  - `getBalance`
  - `insertMoney`
  - `printTicket`

Accessor methods

- Methods implement the behaviour of objects.
- Accessors provide information about an object.
- Methods have a structure consisting of a header and a body.
- The header defines the method's *signature*.
  `public int getPrice()`
- The body encloses the method's statements.
- Further things in the body: declarations.

Accessor methods

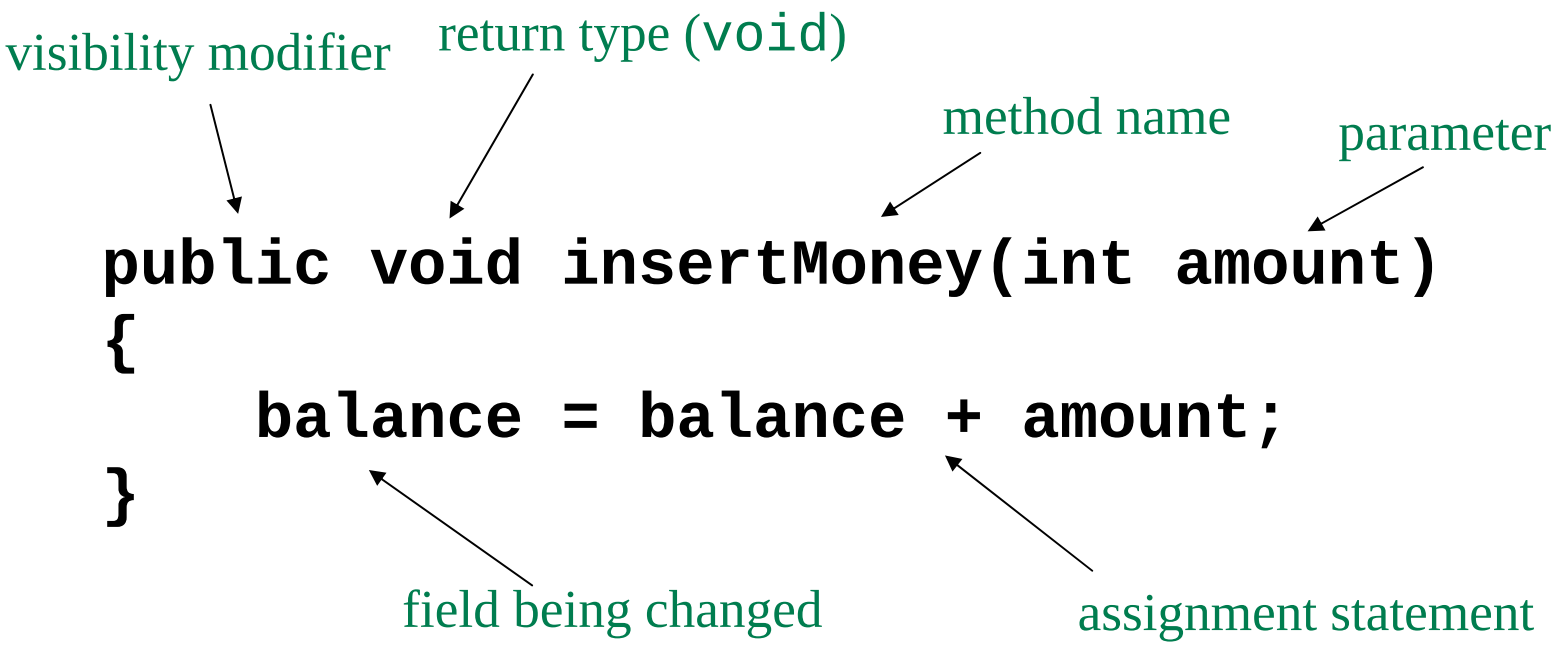


Mutator methods

- 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.
  - Typically receive parameters.

Mutator methods

Note that constructors have no return type!!



Printing from methods

```
public void printTicket()
{
    // Simulate the printing of a ticket.
    System.out.println("####################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("####################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

construct a single string

blank line

Reflecting on the ticket machines

- Their behaviour 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 more sophisticated behaviour.

Making choices – conditional statements

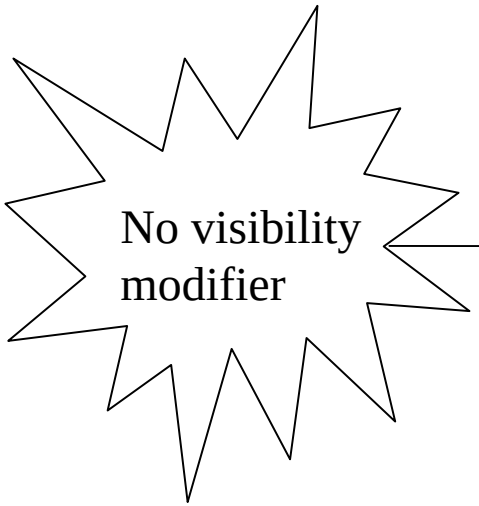
```
public void insertMoney(int amount)
{
    if(amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println("Use a positive amount: " +
                           amount);
    }
}
```

Local variables

- Fields are one sort of variable.
  - They store values through the life of an object.
  - They are accessible throughout the class.
- Methods can include shorter-lived variables.
  - They exist only as long as the method is being executed.
  - They are only accessible from within the method.

Local variables

A local variable, not a field (fields always defined outside methods!)




```
public int refundBalance()
{
    int amountToRefund;
    amountToRefund = balance;
    balance = 0;
    return amountToRefund;
}
```

- lifetime = lifetime of method execution!
- local variables also possible in constructors!

Review (I)

- Class bodies contain fields, constructors and methods.
- Fields store values that determine an object's state.
- Constructors initialize objects.
- Methods implement the behavior of objects.

Review (II)

- Fields, parameters and local variables are all variables.
- Fields persist for the lifetime of an object.
- Parameters are used to receive values into a constructor or method.
- Local variables are used for short-lived temporary storage.

Review (III)

- Objects can make decisions via conditional (if) statements.
- A true or false test allows one of two alternative courses of actions to be taken.