Lecture 2
Chapter 2: Understanding class definitions

Main concepts to be covered

- fields
- constructors
- methods
- parameters [C]
- assignment statements [C]
- conditional statements [C]

[C] ← this symbol means Java works the same way as the C programming language

Naïve-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?

Basic class structure

The outer part of TicketMachine

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

public class ClassName
{
    Fields
    Constructors
    Methods
}
```

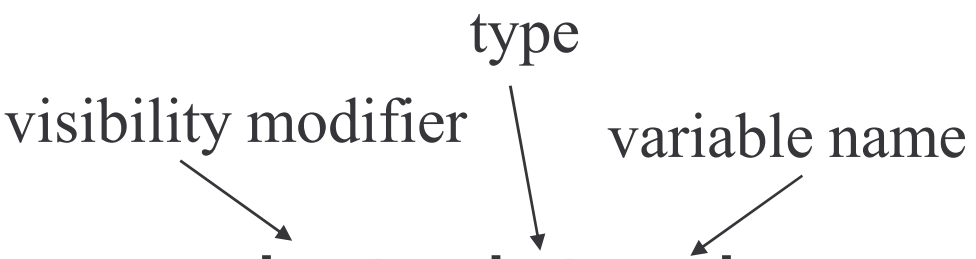
The contents of a class

Fields

- Fields store values for an object
- They are also known as instance variables
- Use BlueJ's *Inspect* option to view an object's fields
- Fields define the state of an object
- Note class names begin with an upper case letter, field names with lower case

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

    Constructor and methods
      omitted.
}
```



Visibility modifiers

- Public fields and methods can be accessed from anywhere
- Private ones can only be accessed within the same class
- Private allows us to hide the implementation of a class from the outside world
- Fields should usually be private, constructors usually public, and methods public only if necessary
- Also called access modifiers
- See section 5.11 of text
- Java also has protected and package visibility modifiers (chapter 9)

Constructors

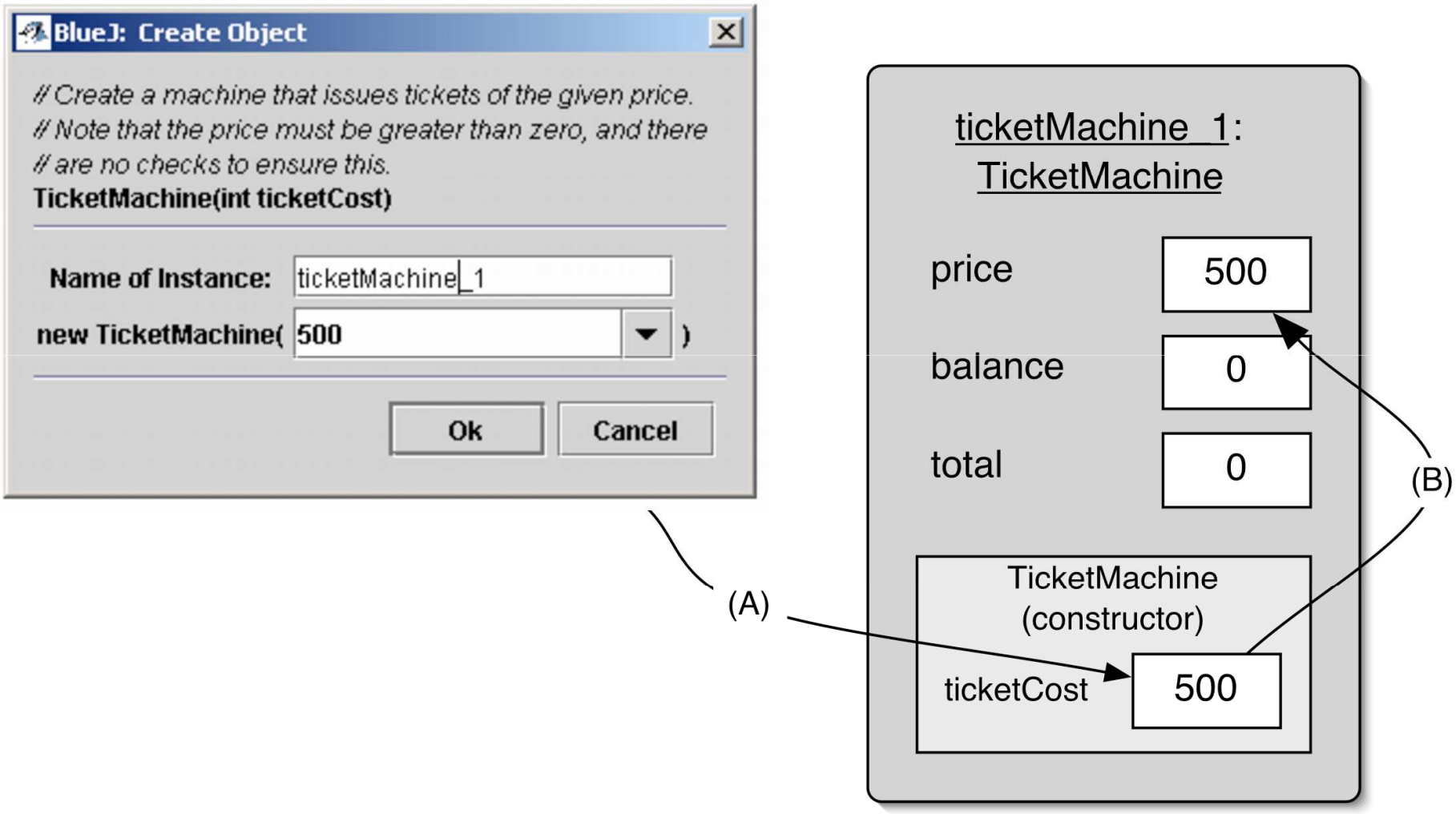
- Special kind of method that initialises an object
- They have the same name as their class
- They store initial values in the fields
- Often receive external parameter values to put in fields

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

Defines the class. We won't show it from now on

Constructor

Passing data via parameters



Overloading constructors

- Constructors can have the same name as long as they have different parameter lists
- This is called overloading
- It gives a way of doing the same thing using different arguments

```
public TicketMachine() {…
public TicketMachine(int ticketCost) {…
```

- Methods can also be overloaded
- See section 3.10 in text

Scope

- A variable can only be used within its *scope*
- Scope of a parameter is the method it occurs in
- Scope of a field is at least the entire class
  - visibility modifiers determine the full scope of a field
  - public fields have global scope
  - using the most restricted scope you reasonably can helps avoid bugs
    - Use a local variable instead of a field, if you can
    - Use private visibility modifier when you can
- Notice the scope differences for the `ticketCost` parameter and `price` field on earlier slides

Lifetime

- A parameter's lifetime ends when its method finishes executing
  - The data in a parameter is lost when the method finishes
- A field's lifetime ends when its object is destroyed
  - The data in a field is lost when this happens
  - Java's garbage collector automatically destroys objects when they're no longer used

Assignment [C]

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

  variable = *expression;*

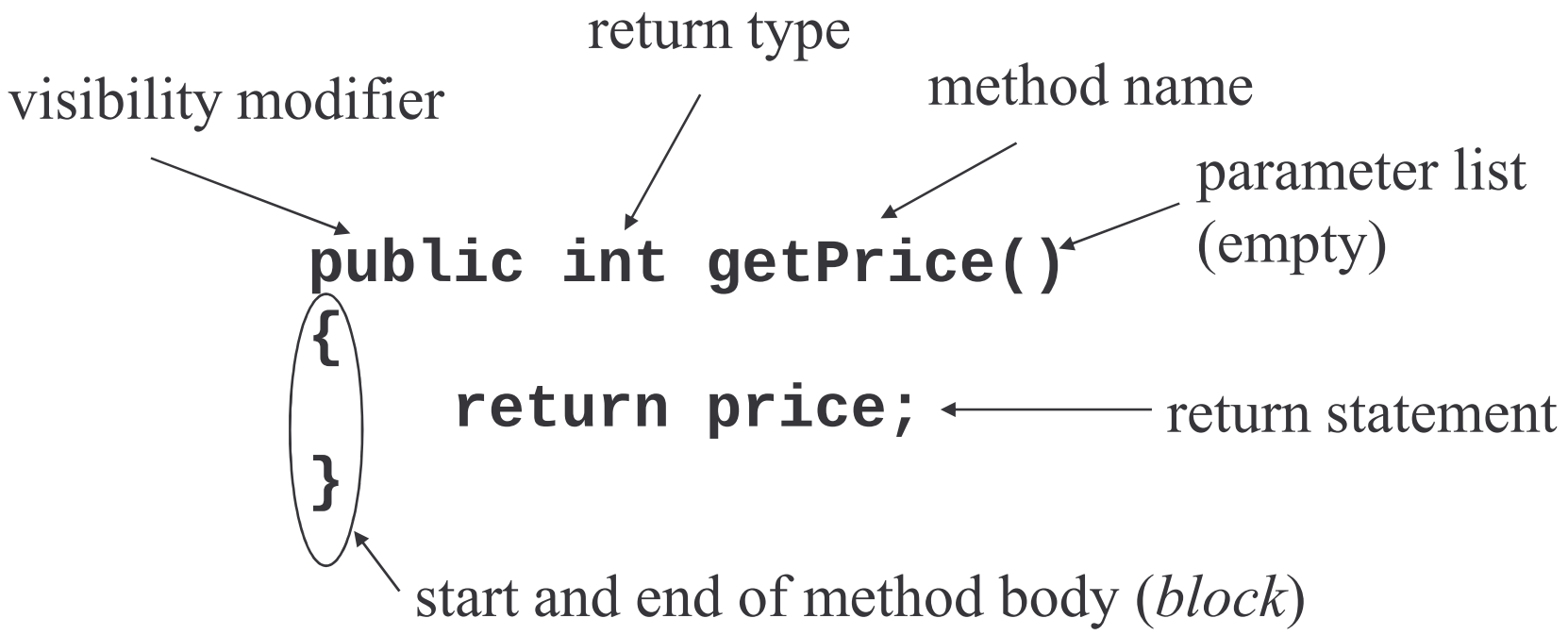
  ```
  price = ticketCost;
  ```

- A variable stores a single value, so any previous value is lost.

Methods

- Methods implement the behavior of objects.
- 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 within curly brackets { }

Accessor methods

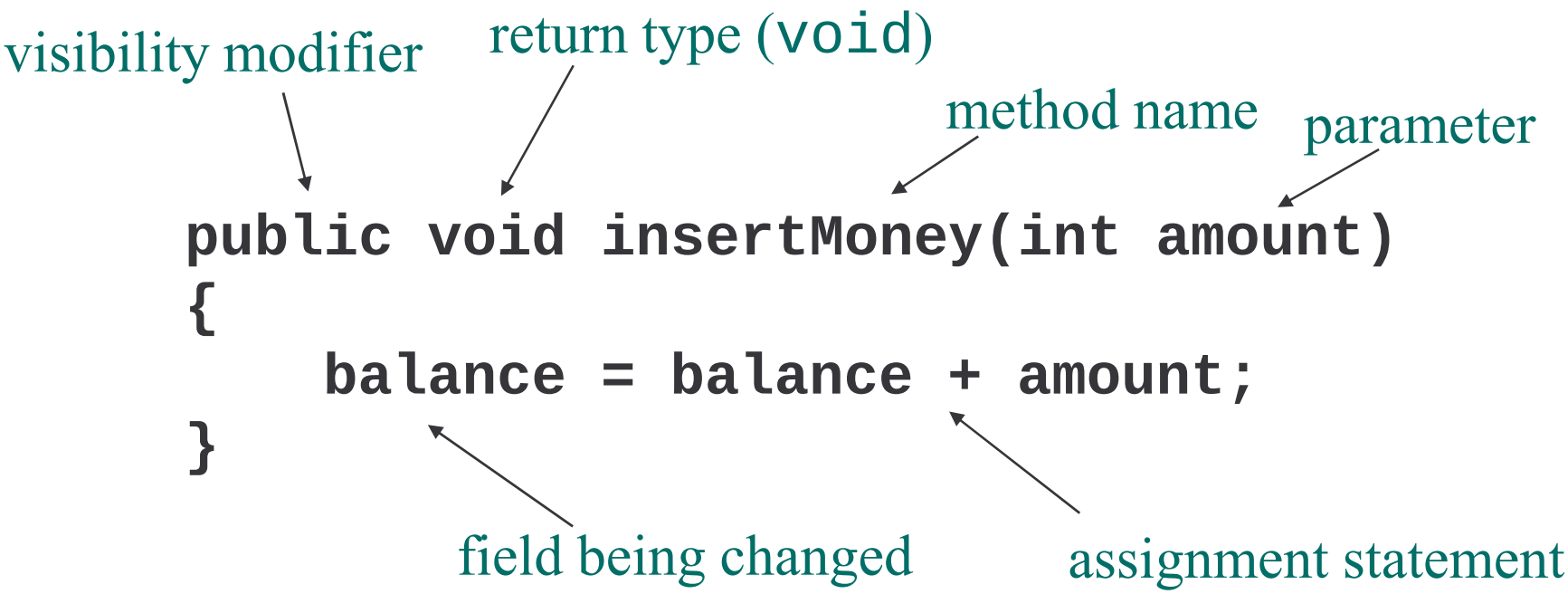


- Accessors provide information about an object.
- Also called “get methods”

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
- Also called "set methods"

Mutator methods



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;
}
```

Notes on printing

- + operator concatenates (joins) strings
- Any other type joined to a string is first converted to a string

Reflecting on the ticket machines

- Imagine you wrote TicketMachine
- How could someone else misuse it?
- Several deficiencies:
  - No checks on the amounts entered
    - E.g. entering -10
  - No refunds
  - No checks for a sensible initialization
    - E.g. ticketCost of -10
- We need more sophisticated behavior

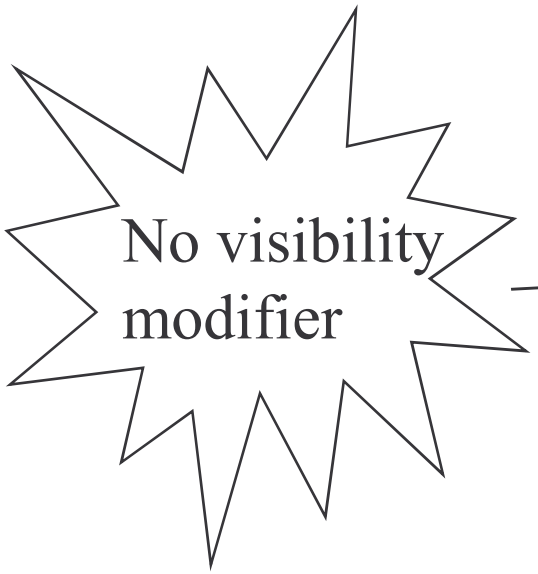
Validating a Parameter

```
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: local variables.
  - They exist only as long as the method is being executed.
  - They are only accessible from within the method.
  - Parameters are actually a special kind of local variable whose value is passed into method.

Local variables



A local variable

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

Review

- 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

- 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
- Fields should usually be private and only accessible by accessor and mutator methods