

Understanding class definitions

Looking inside classes

Objects and classes

- **objects**
 - represent ‘things’ from the real world, or from some problem domain (example: “the red car down there in the car park”)
- **classes**
 - represent all objects of a kind (example: “car”)

Methods and parameters

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

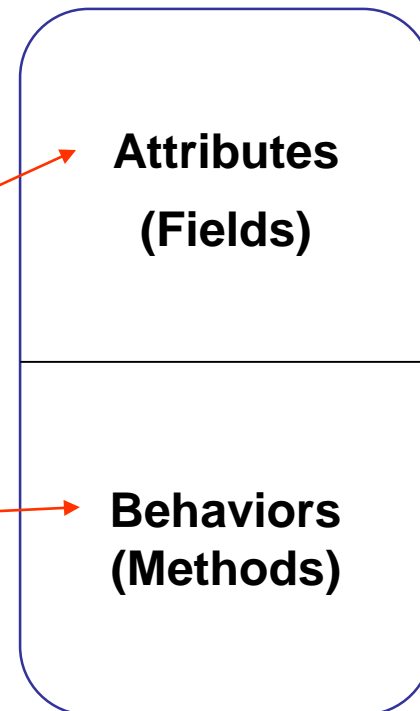
Other observations

- Many *instances* can be created from a single class.
- An *object* has *attributes*: values stored in *fields*.
- The *class* defines what fields an object has, but each object stores its own set of values (the *state* of the object).

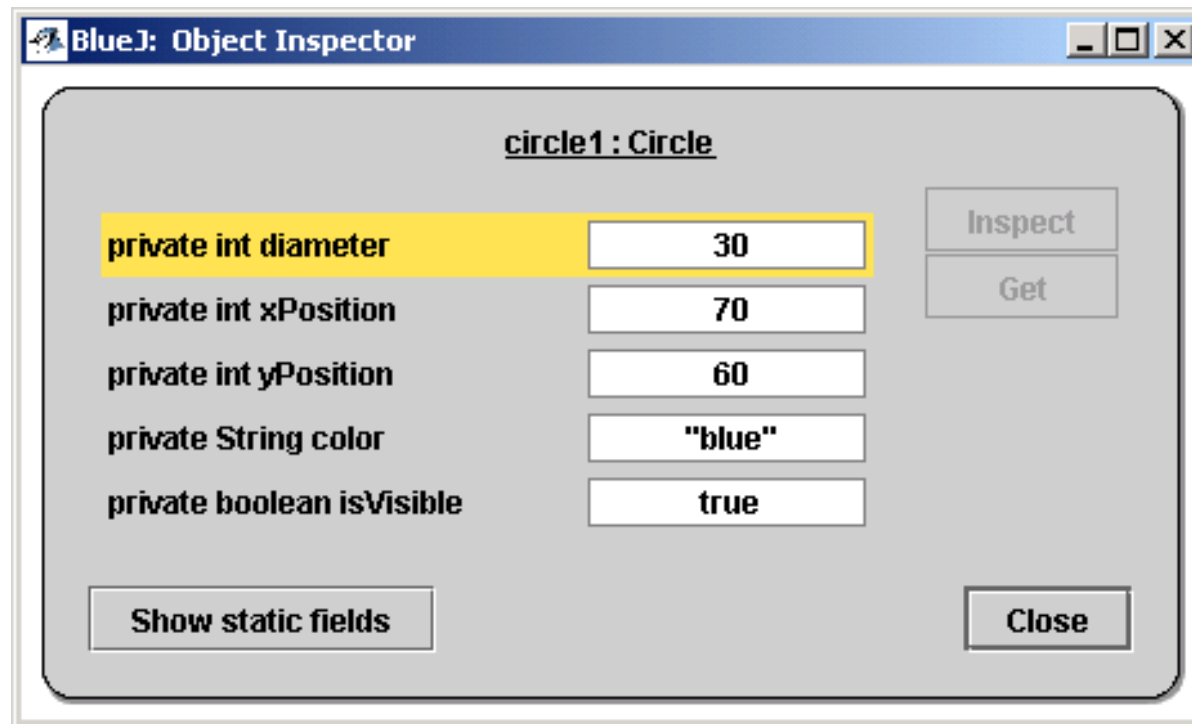
Class

- A blueprint for objects of a particular type
- Defines the structure (number, types) of the attributes
- Defines available behaviors of its objects

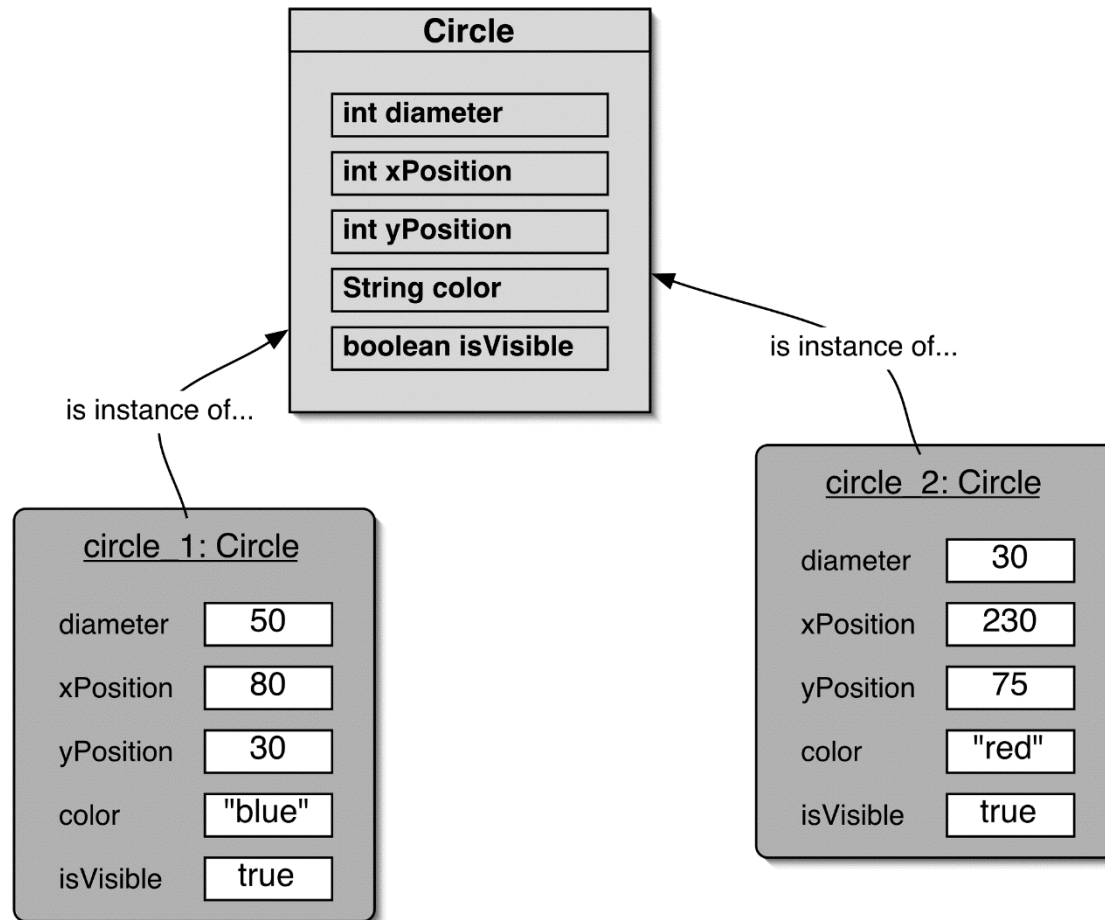
Object



State



Two circle objects



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?

Basic class structure

The outer wrapper
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**.
- Fields define the **state** of an object.

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

    Further details omitted.
}
```

visibility modifier type variable name

↓ ↓ ↓

private int price;

Visibility

- **Private** members
 - Can be accessed only by instances of same class
 - Provide concrete implementation / representation
- **Public** members
 - Can be accessed by any object
 - Provide abstract view (client-side)
- **Protected** members
 - Can be accessed by instances of the same class and its subclasses

Declaration with an access modifier

- Each class declaration that begins with the access modifier **public** must be stored in a file that has **exactly the same name** as the class and ends with the **.java** file-name extension.

Constructors

- **Constructors** initialize an object.
- They have **the same name** as their class.
- They store **initial** values into the fields.
- They often receive external parameter values for this.

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

Constructors (cont.)

- A constructor is a procedure for creating objects of the class.
- Keyword **new** requests memory from the system to store an object, then calls the corresponding class's constructor to initialize the object.
- A constructor often **initializes** an object's fields.
- Constructors do not have a **return type** (not even void) and they do not return a value.
- **All constructors** in a class have the same name – **the name of the class**.
- Constructors may take **parameters**.

Constructors (cont.)

- If a class has more than one constructor, they must have **different** numbers and/or types of parameters.
- Programmers often provide a “**no-args**” constructor that takes no parameters (a.k.a. *arguments*).
- If a programmer does not define any constructors, Java provides one default (no-args) constructor, which allocates memory and sets fields to the default values.

Constructors (cont.)

Code Example...

- 1) Naïve-TicketMachine
- 2) Just Modified version

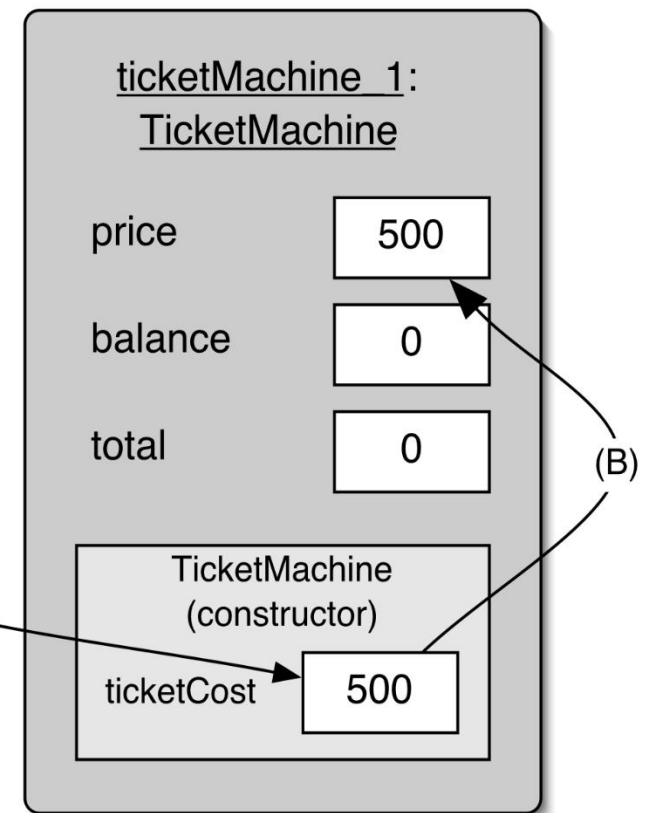
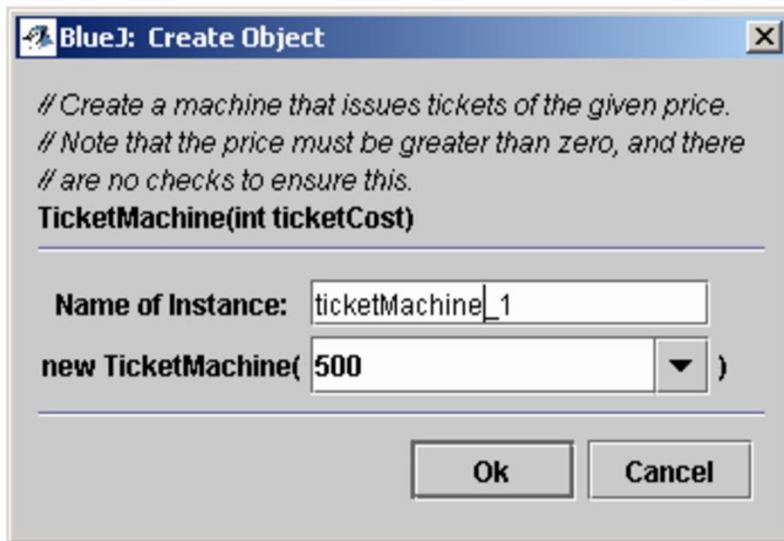
Constructors (cont.)

A **nasty** bug:

```
public class MyClass
{
    ...
    // Constructor:
    public void MyClass (...)
    {
        ...
    }
    ...
}
```

Compiles fine, but the compiler thinks this is a method and uses **MyClass**'s default no-args constructor instead.

Passing data via parameters



Accessor methods

- Methods implement the **behavior** 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**.

Accessor methods

The diagram illustrates the components of a Java accessor method. The code is: `public int getPrice() { return price; }`. Annotations with arrows point to specific parts: 'visibility modifier' points to 'public'; 'return type' points to 'int'; 'method name' points to 'getPrice'; 'parameter list (empty)' points to '()'; 'return statement' points to 'return price;'; and 'start and end of method body (block)' points to the curly braces '{ }' which are enclosed in a green oval.

visibility modifier

return type

method name

parameter list (empty)

return statement

start and end of method body (block)

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


Mutator methods

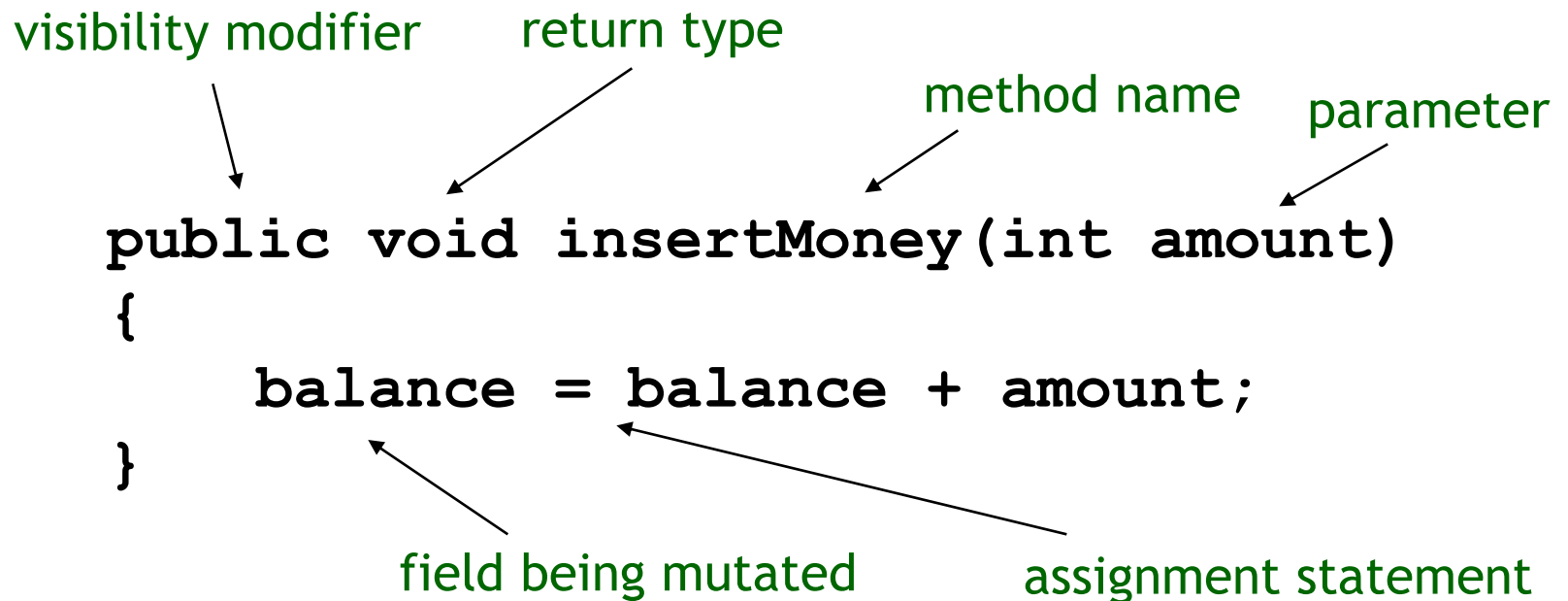
- Have a similar method structure: header and body.
- Used to *mutate* (i.e., change) an object's state.
- Achieved through changing the value of one or more fields.
 - Typically contain assignment statements.
 - Typically receive parameters.

Mutator methods

visibility modifier return type method name parameter

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

field being mutated assignment statement

A diagram illustrating the components of a Java mutator method. The code snippet is: `public void insertMoney(int amount) { balance = balance + amount; }`. Annotations with arrows point to specific parts: 'visibility modifier' points to 'public'; 'return type' points to 'void'; 'method name' points to 'insertMoney'; 'parameter' points to 'int amount'; 'field being mutated' points to the first 'balance' in the assignment statement; and 'assignment statement' points to the entire line 'balance = balance + amount;'.

The diagram illustrates the components of a Java mutator method. The code snippet is: `public void insertMoney(int amount) { balance = balance + amount; }`. Annotations with arrows point to specific parts: 'visibility modifier' points to 'public'; 'return type' points to 'void'; 'method name' points to 'insertMoney'; 'parameter' points to 'int amount'; 'field being mutated' points to the first 'balance' in the assignment statement; and 'assignment statement' points to the entire line 'balance = balance + amount;'. The labels 'field being mutated' and 'assignment statement' are positioned below the code, with arrows pointing to the first 'balance' and the entire assignment line respectively.

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;
}
```

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 more sophisticated behavior.

Making choices

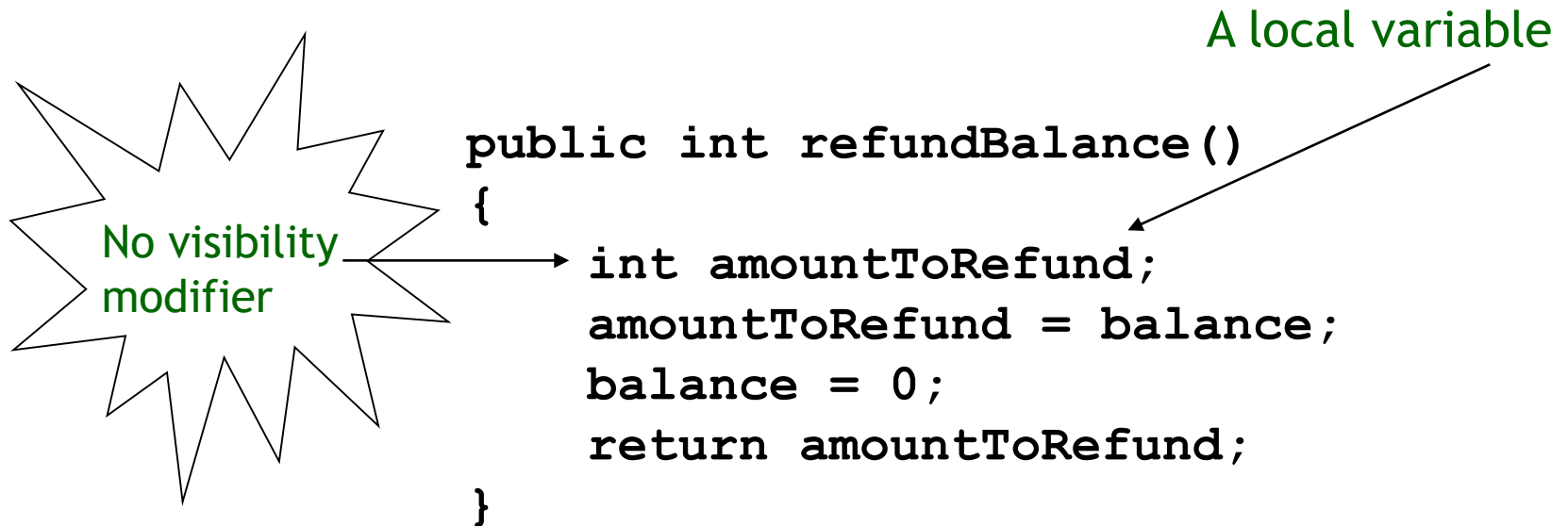
```
public void insertMoney(int amount)
{
    if(amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println("Use a positive amount: " +
                           amount);
    }
}
```

Now Better-TicketMachine

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



Again Main.java last part

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.

شعر امروز

عمریست که جان بنده بی خویشتن است
و انگشت‌نمای عالمی مرد و زن است

برخاستن از جان و جهان مشکل نیست
مشکل ز سر کوی تو برخاستن است