Understanding class definitions

Looking inside classes

Fundamental concepts

- object
- class
- method
- parameter
- data type

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.

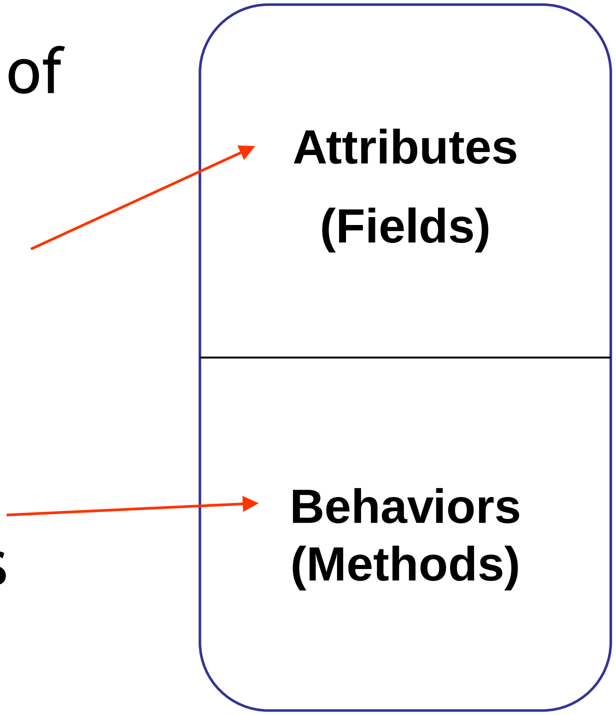
Other observations

- 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

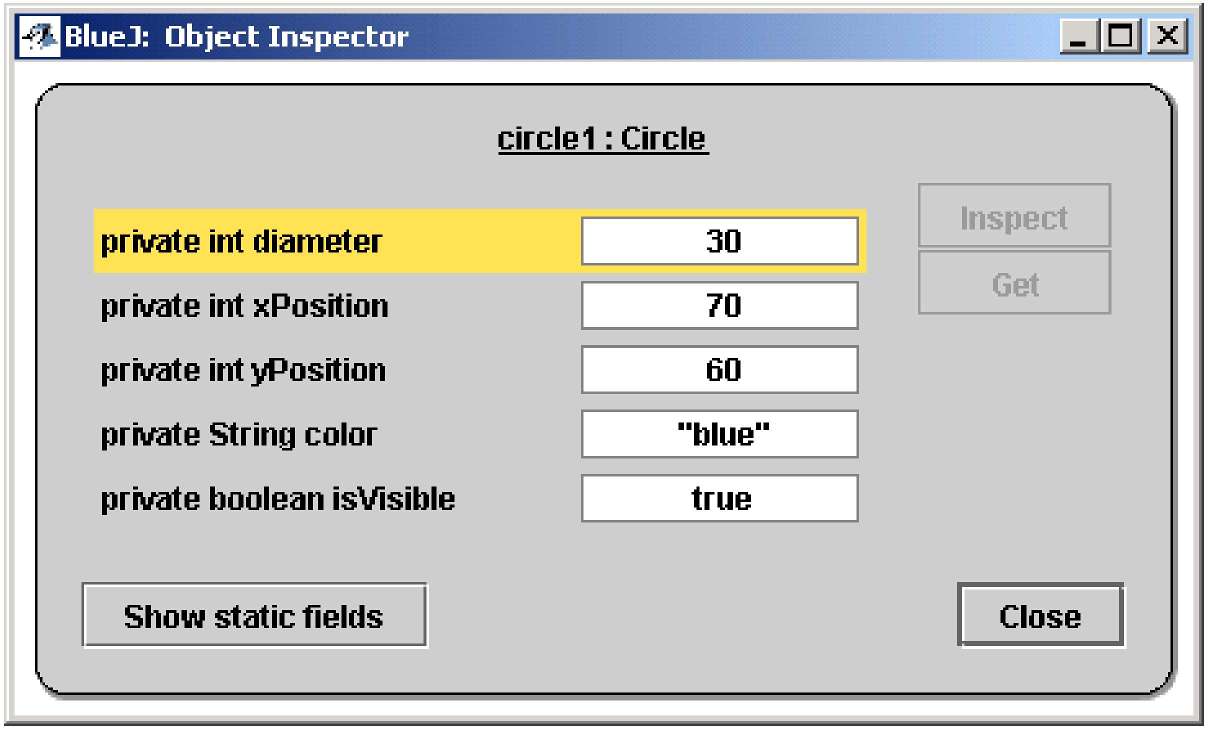


Attributes

(Fields)

Behaviors
(Methods)

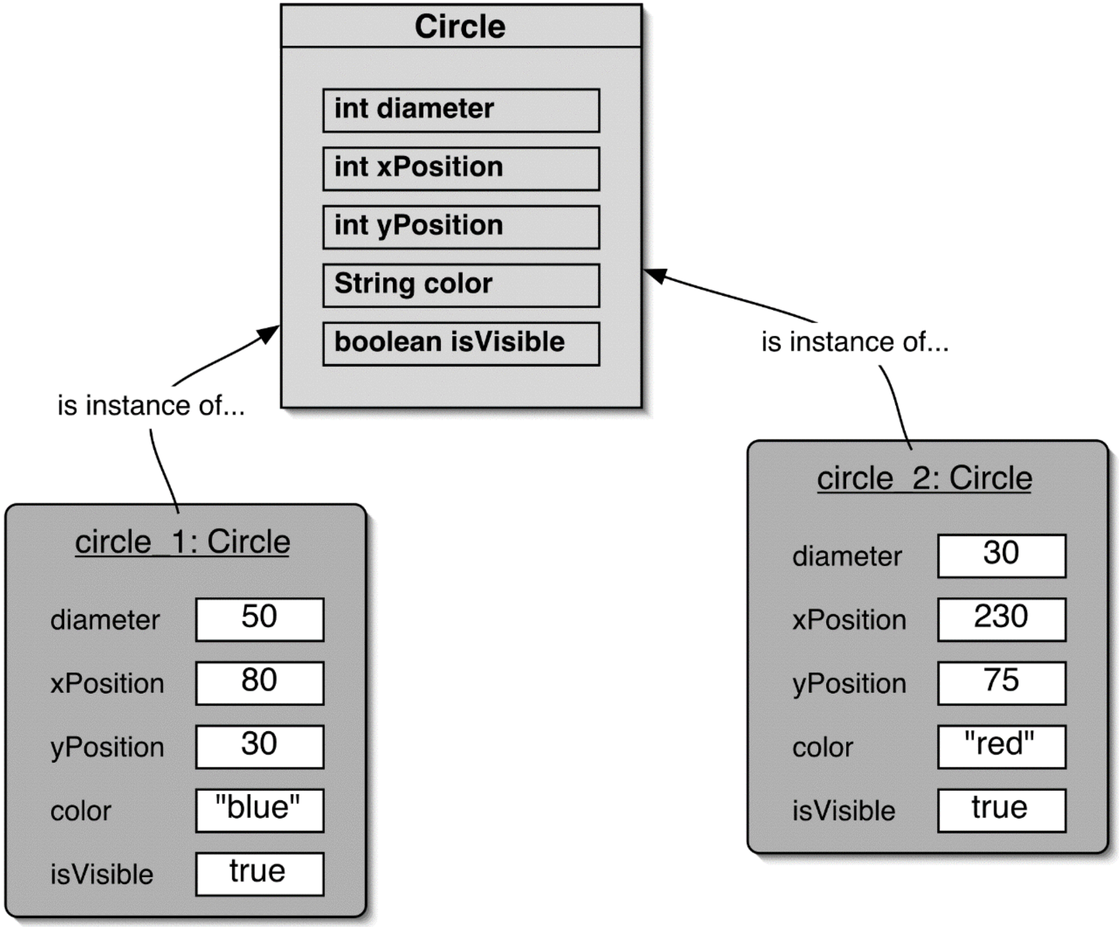
State



circle1 : Circle

private int diameter	30
private int xPosition	70
private int yPosition	60
private String color	"blue"
private boolean isVisible	true

Two circle objects



Source code

- Each class has source code (Java code) associated with it that defines its details (fields and methods).

Main concepts to be covered

- fields
- constructors
- methods
- parameters

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
}
```

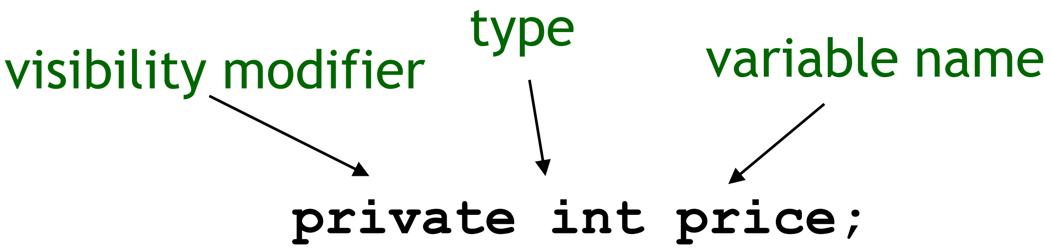
The contents of a class

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.
}
```



type

visibility modifier

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 (without Main.java)

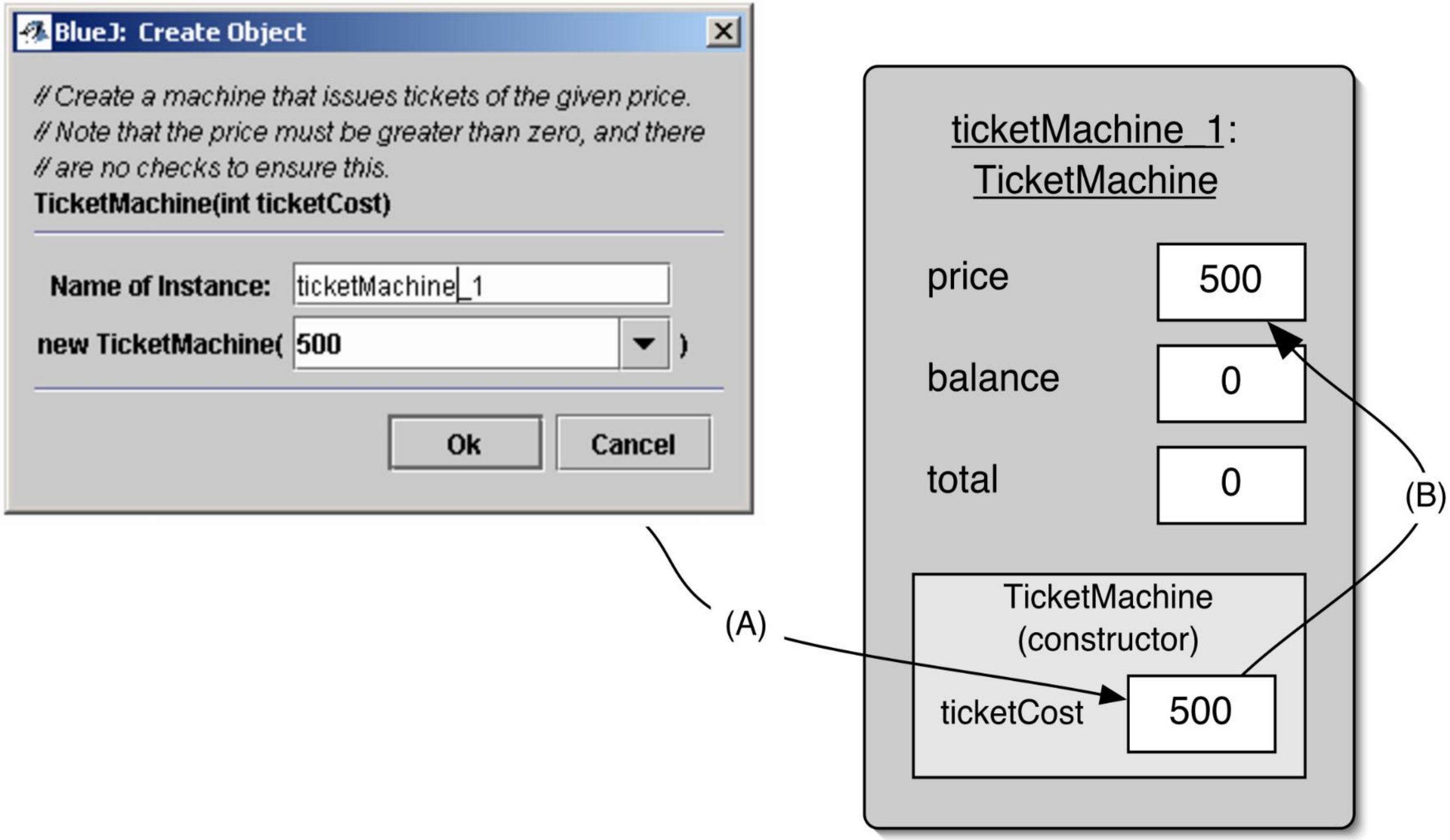
Constructors (cont.)

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



BlueJ: Create Object

```
// Create a machine that issues tickets of the given price.
// Note that the price must be greater than zero, and there
// are no checks to ensure this.
TicketMachine(int ticketCost)
```

Name of Instance: ticketMachine_1

new TicketMachine(500)

new

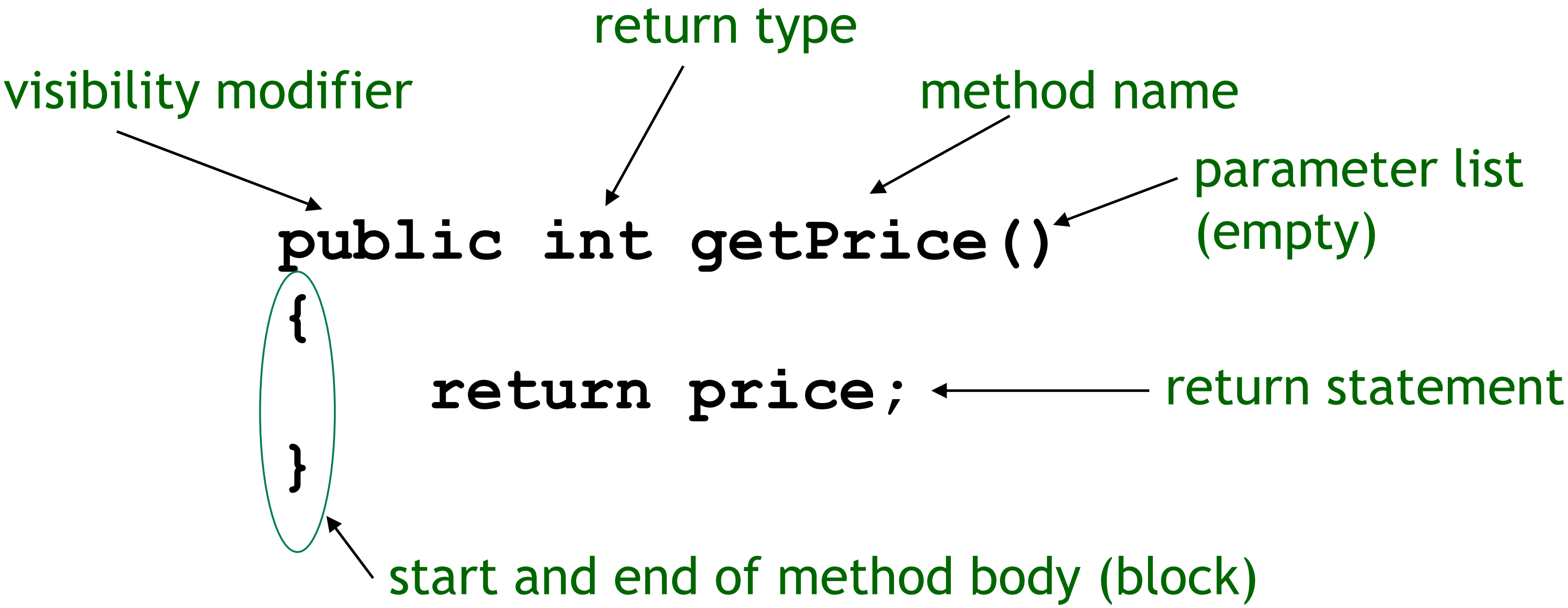
- Constructors are invoked using the operator new.
- Code Example…
  (Now Main.java before local vars)

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



return type

visibility modifier

method name

parameter list (empty)

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

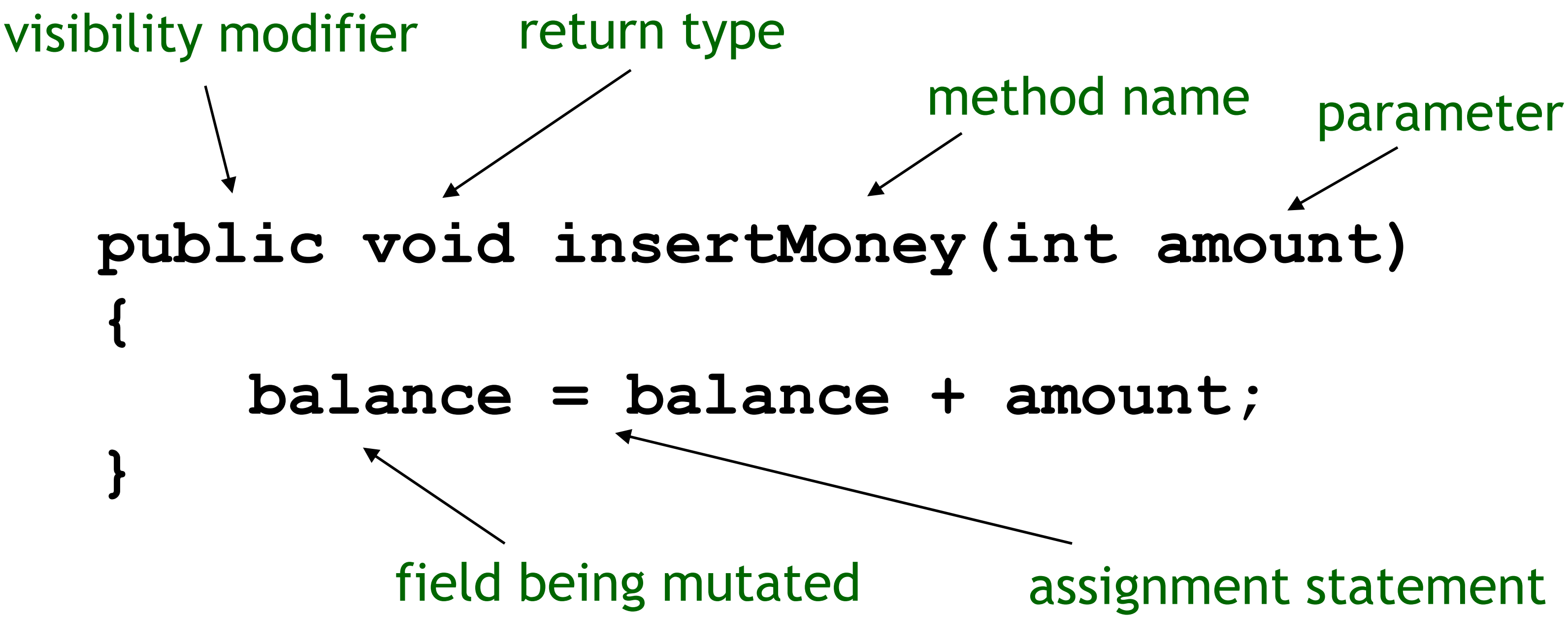
return statement

start and end of method body (block)

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

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

A local variable

No visibility modifier

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

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.