Understanding class definitions

Looking inside classes
(based on lecture slides by Barnes and Kölling)
Main Concepts

- fields
- constructors
- methods
- parameters
- assignment statements
Ticket Machines (an external view)

- Exploring the behaviour 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
Ticket machines (an internal view)

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

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

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

- Fields store values for an object.
- They are also known as instance variables.
- Use the *Inspect* option to view an object’s fields.
- Fields define the state of an object.

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

    Further details omitted.
}
```
Constructors initialise 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.

```java
public TicketMachine(int ticketCost) {
    price = ticketCost;
    balance = 0;
    total = 0;
}
```
Passing data via parameters

```
// 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 )
```

```
ticketMachine_1:
TicketMachine

<table>
<thead>
<tr>
<th>Price</th>
<th>Balance</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

(A) TicketMachine (constructor)
(B) ticketCost = 500
Assignment

- Values are stored into fields (and other variables) via assignment statements:
  
  \[ \text{variable} = \text{expression}; \]
  
  \[ \text{price} = \text{ticketCost}; \]

- A variable stores a single value, so any previous value is lost.
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.
  ```java
  public int getPrice()
  ```
- The body encloses the method’s statements.
public int getPrice() {
    return price;
}
What is wrong here? (there are five errors!)

```java
test
public class CokeMachine {
    private int price;

    public CokeMachine() {
        price = 300;
    }

    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.
public void insertMoney(int amount) {
    balance = balance + amount;
}
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;
}
String concatenation

- $4 + 5 \quad \rightarrow \quad \text{overloading} \quad 9$
- "wind" + "ow" -> "window"
- "Result: " + 6 -> "Result: 6"
- "# " + price + " cents" -> "# 500 cents"
Quiz

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

  11hello

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

  hello56
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 initialisation.

- How can we do better?
  - We need more sophisticated behaviour.
Making choices

```java
public void insertMoney(int amount) {
    if(amount > 0) {
        balance = balance + amount;
    } else {
        System.out.println("Use a positive amount: "+ amount);
    }
}
```
How do we write 'refundBalance'?
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.
Scope and life time

- The scope of a local variable is the block it is declared in.
- The lifetime of a local variable is the time of execution of the block it is declared in.
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 initialise objects.

• Methods implement the behaviour 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.
Review

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