Software Development (CS2500)

Lecture 3: Introduction to Objects and Classes

M. R. C. van Dongen

September 27, 2013

Objects and Classes

- Programmers construct their `Java` program from *objects*.
- Similar to a builder building a house from parts:
  - Doors;
  - Windows;
  - Walls;
  - ...
- Each part has its own function.
- The parts work together to form the house:
  - The house is the sum of the parts.
- The builder doesn't have to construct the parts.
- All he does is composing them.

Using Objects

- Objects are the first citizens of Java programs.
- You make an object work by calling its methods.
- Each method is a sequence of instructions.
- You can call a method even if you don't know its instructions.

Java

```
System.out.println( "Hello world!" );
```

- Each method provides a service.
  - The method performs the service if you call the method.
- Different methods may provide differen services.
  - Draw a picture;
  - Print text;
  - Set up a connection with a different computer;
  - Compute something and return it;
  - ...

Classes

- Each object belongs to a unique class.
- Different objects may belong to different classes.
  - `System.out`
  - `"Hello world!"`
- An object that belongs to a class is called an *instance* of the class.
- A class may have more than one instance:
  - `"Hello world!"`
  - `"What's up Doc?"`
  - ...

Classes

- Each object belongs to a unique class.
- Different objects may belong to different classes.
  - `System.out`
  - `"Hello world!"`
- An object that belongs to a class is called an *instance* of the class.
- A class may have more than one instance:
  - `"Hello world!"`
  - `"What's up Doc?"`
  - ...

Classes

- Each object belongs to a unique class.
- Different objects may belong to different classes.
  - `System.out`
  - `"Hello world!"`
- An object that belongs to a class is called an *instance* of the class.
- A class may have more than one instance:
  - `"Hello world!"`
  - `"What's up Doc?"`
  - `...`

Classes

- Each object belongs to a unique class.
- Different objects may belong to different classes.
  - `System.out`
  - `"Hello world!"`
- An object that belongs to a class is called an *instance* of the class.
- A class may have more than one instance:
  - `"Hello world!"`
  - `"What's up Doc?"`
  - `...`

Classes (Continued)

- Each class has its own *Application Programming Interface* (API).
- The API describes how to use the class:
  - The names of the methods;
  - The types of the arguments;
  - The purpose of the arguments;
  - The return value;
  - Side effects;
  - ...
- The API defines a common protocol:

Java

```
System.out.println( "Hello world!" );
System.err.println( "Fatal error." );
```

- Different classes may have different APIs.
  - E.g. an instance of the `String` class cannot print.

Don't Try This at Home

```
"Hello world!".println( "What's up Doc?" );
```

Variables

- Most programs require computations.
  - Add 13% VAT to the price;
  - Add 2 penalty points;
  - Determine the maximum input value;
  - ...
- A single computation may require many sub-computations.
- You store the results of a compuation in a *variable*.
- A variable has several properties:
  - A name;
  - A memory location to store its value;
  - Its current value.
- To change a variable's value, you *assign* it a new value.

Java

```
⟨variable's name⟩ = ⟨expression that determines the value⟩;
```

Variables (Continued)

- Before you can use a variable, you must *declare* it.
- A variable declaration determines:
  - The variable's name;
  - The variable's type (the kind of its values);

Java

```
int counter;
double interest;
```

- A variable declaration may also determine the initial value;

Java

```
String greetings = "Hello world!";
```

Assignment and Equality

- In mathematics you use = for equality.
- In Java you use = for assignment.

Assignment and Equality

- In mathematics you use = for equality.
- In Java you use = for assignment.
- But assignment and equality are not the same.

Assignment and Equality

- In mathematics you use = for equality.
- In Java you use = for assignment.
- But assignment and equality are not the same.
- The symbols are the “same” but they don’t mean the same.

Assignment and Equality

- In mathematics you use = for equality.
- In Java you use = for assignment.
- But assignment and equality are not the same.
- The symbols are the "same" but they don't mean the same.
- Mathematical equality is commutative: if $a = b$, then $b = a$.

Assignment and Equality

- In mathematics you use = for equality.
- In Java you use = for assignment.
- But assignment and equality are not the same.
- The symbols are the "same" but they don't mean the same.
- Mathematical equality is commutative: if $a = b$, then $b = a$.
- You can't write the following in Java:

Don't Try This at Home

```
1 = a; // ?
```

Assignment and Equality

- In mathematics you use = for equality.
- In Java you use = for assignment.
- But assignment and equality are not the same.
- The symbols are the "same" but they don't mean the same.
- Mathematical equality is commutative: if $a = b$, then $b = a$.
- You can't write the following in Java:

Don't Try This at Home

```
1 = a; // ?
```

- In mathematics $a = a + 1$ is impossible.

Assignment and Equality

- In mathematics you use = for equality.
- In Java you use = for assignment.
- But assignment and equality are not the same.
- The symbols are the "same" but they don't mean the same.
- Mathematical equality is commutative: if $a = b$, then $b = a$.
- You can't write the following in Java:

Don't Try This at Home

```
1 = a; // ?
```

- In mathematics $a = a + 1$ is impossible.
- In Java you can write:

Java

```
counter = counter + 1;
```

Types

- Java has different numeric types.
  - whole numbers
    - `byte`;
    - `short`;
    - `int`;
    - `long`.
  - floating point
    - `float`;
    - `double`.
- For whole numbers, `int` is usually a good choice.
- For floating point numbers, use `double`.

Operations on Numbers

unary plus ⟨assignee⟩ = + ⟨operand⟩;
unary minus ⟨assignee⟩ = - ⟨operand⟩;
adding ⟨assignee⟩ = ⟨operand #1⟩ + ⟨operand #2⟩;
subtracting ⟨assignee⟩ = ⟨operand #1⟩ - ⟨operand #2⟩;
multiplying ⟨assignee⟩ = ⟨operand #1⟩ * ⟨operand #2⟩;
dividing ⟨assignee⟩ = ⟨operand #1⟩ / ⟨operand #2⟩;
...

- Multiplicative operators bind more tightly:
  - `a * b + c` equals `c + a * b` equals `(a * b) + c`.
  - `a / b + c` equals `c + a / b` equals `(a / b) + c`.

Constant Variables

- A *constant* (variable) can only be assigned a value once.
- You declare a constant by adding the keyword `final`.

Java

```
final int ANSWER = 42;
```

- Making a variable constant is a form of documentation.
- It lets the compiler help you detect logic errors:

Java

```
final int ACCELLERATION = 9.8;
...
ACCELLERATION = 9.9;
```

Using Variables in Methods

- You cannot use an unassigned variable in a method.

Don't Try This at Home

```
int number;
int square = number * number;
```

Comments

- A *comment* is text that is ignored by the compiler.
- Comments have several purposes:
  - They describe the purpose of a variable/method.
  - They describe a relationship between two or more variables.
    - This is called an invariant.
  - They are used to create API documentation.
- You should always document your programs.

One Line Comments

Java

```
// number of centimetres per inch
final double CENTIMETRES_PER_INCH = 2.56;
```

Multi-Line Line Comments

Java

```
/* Encrypted user password.
 * Use the changePassword( ) method to change the password.
 */
String password;
```

JavaDoc Comments

Java

```
/**
 * More about these in another lecture.
 */
```

Variable Names

- Use names that are meaningful.
- The name should describe the variable's purpose.
- By convention aach variable name should be a noun.

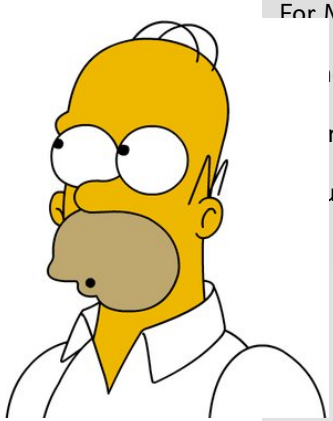
non-constant
- Each name should start with a lowercase letter.
- The rest should be letters and digits.
- At word boundaries, you use an uppercase letter.
- All other letters should be lowercase.
- E.g. `sum`, `currentColour`, ...

constant
- Names are sequences of words & underscores.
- Each word is spelt with uppercase letters.
- At word boundaries, you use an underscore.
- E.g. `CENT`, `CENTIMETRES_PER_INCH`, ...

Choosing Variable Names

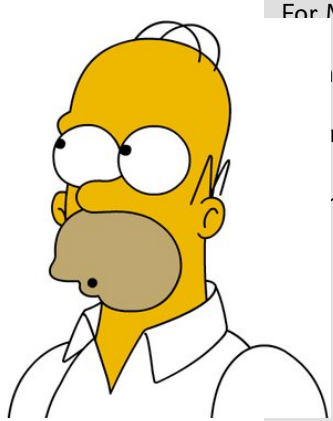
Choosing Variable Names

- Variable names should be descriptive.



Choosing Variable Names

- Variable names should be descriptive.
- This is a form of documentation:
  - It helps you remember what the variable does.
  - It helps others understand the purpose of the variable.



Choosing Variable Names

- Variable names should be descriptive.
- This is a form of documentation:
  - It helps you remember what the variable does.
  - It helps others understand the purpose of the variable.

Choosing Variable Names

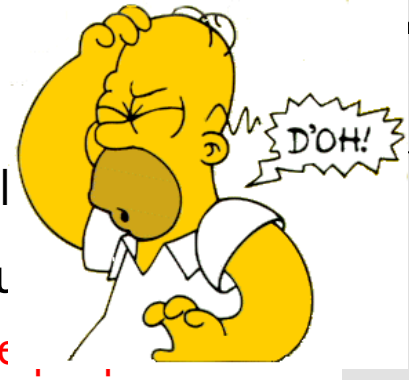
- Variable names should be descriptive.
- This is a form of documentation:
  - It helps you remember what the variable does.
  - It helps others understand the purpose of the variable.

Choosing Variable Names

- Variable names should be descriptive.
- This is a form of documentation:
  - It helps you remember what the variable does.
  - It helps others understand the purpose of the variable.
- Choosing a good name helps you understand the purp

Choosing Variable Names

- Variable names should be descriptive.
- This is a form of documentation:
  - It helps you remember what the variable does.
  - It helps others understand the purpose of the variabl
- Choosing a good name helps you understand the pu
  - If you can't find a good name, do you really know the



Working with Objects

- Before you can use an object, you must initialise (create) it.
- To initialise an object, you *construct* it.
- There may be different ways to construct an object.

Java

```
final Rectangle bar = new Rectangle( x, y, width, height );
```

Constructing the Rectangle

Java

```
Rectangle bar = new Rectangle( x, y, width, height )
```

1. The new operator creates space for the Rectangle object;
2. It uses the arguments of the constructor to initialise the object;
3. It returns a *reference* to the object;
4. The reference is assigned to the object reference value bar.
5. The reference may be used to call the object's methods.

Method Declarations

- To declare a method you provide:
  - The name of the method;
  - The return type;
  - The names and types of the formal parameters;
  - The types of the formal parameters.

Java

```
public int getWidth( ) { /* Implementation omitted. */ }
```

- You use `void` for a method without return value.

Java

```
public void println( String output ) { /* Implementation omitted. */ }
```

- If the argument types are different, the names may overlap.
  - This is called *overloading:*

Java

```
public void println( int output ) { /* Implementation omitted. */ }
```

Accessor and Mutator Methods

- A method that returns information about an object without modifying the object is called an *accessor method.*
  - `double width = rectangle.getwidth( );`
- A method that modifies an object's instance variables is called a *mutator method.*
  - `rectangle.setWidth( 4.0 );`

The API Documentation

- Intermezzo.

Implementing a Tally Counter Class

- Let's implement a tally counter object class.
- The name of the class should be a noun.
  - The name should start with an uppercase letter.
  - The name should continue with letters and digits.
  - At each word boundary, you use an uppercase letter.
  - All other letters should be lowercase.
  - The name should describe the instances of the class.
  - For example, `StringBuilder`, `FullAdder`, ...

State and Behaviour

- Let's use `Counter` for our class name.
- How do we implement the class?
- We must determine what the `Counter` instances do and know.
- What the instance does is called its *behaviour*.
  - Object behaviour is implemented as *instance methods*.
- What the instance knows is called its *state*.
  - Object state is implemented as *instance variables*.

Behaviour

What Should `Counter` Objects Do?

- Compute the next counter value.
  - `public void incrementValue( )`
- Give us the current counter value.
  - `public int getValue( )`

What Should `Counter` Objects Know?

- Their counter value.
  - `private int value;`

The Class

Class Definition

Java

```
// Class for representing tally counter objects.
public class Counter {
    // The current tally counter value.
    private int value;

    // Returns the current counter value.
    public int getValue( ) {
        return value;
    }

    // Increment the counter value.
    public void incrementValue( ) {
        value = value + 1;
    }
}
```

The Class

Instance Attribute Declaration

Java

```
// Class for representing tally counter objects.
public class Counter {
    // The current tally counter value.
    private int value;

    // Returns the current counter value.
    public int getValue( ) {
        return value;
    }

    // Increment the counter value.
    public void incrementValue( ) {
        value = value + 1;
    }
}
```

The Class

Instance Method Declarations

Java

```
// Class for representing tally counter objects.
public class Counter {
    // The current tally counter value.
    private int value;

    // Returns the current counter value.
    public int getValue( ) {
        return value;
    }

    // Increment the counter value.
    public void incrementValue( ) {
        value = value + 1;
    }
}
```

The Class

Access/Visibility Specifiers

Java

```
// Class for representing tally counter objects.
public class Counter {
    // The current tally counter value.
    private int value;

    // Returns the current counter value.
    public int getValue( ) {
        return value;
    }

    // Increment the counter value.
    public void incrementValue( ) {
        value = value + 1;
    }
}
```

The Class

Types

Java

```
// Class for representing tally counter objects.
public class Counter {
    // The current tally counter value.
    private int value;

    // Returns the current counter value.
    public int getValue( ) {
        return value;
    }

    // Increment the counter value.
    public void incrementValue( ) {
        value = value + 1;
    }
}
```

The Class

Comments

Java

```
// Class for representing tally counter objects.
public class Counter {
    // The current tally counter value.
    private int value;

    // Returns the current counter value.
    public int getValue( ) {
        return value;
    }

    // Increment the counter value.
    public void incrementValue( ) {
        value = value + 1;
    }
}
```

Instance Variables

- Each `Counter` object has its own `value` variable.
- If `tally` is a `Counter` object reference, it is called `tally.value`.
- The `Counter` object owns the variable.
- Different `Counter` objects may have different values for `value`.

Instance Methods

- `Counter` objects can call `Counter` instance methods.
- Calling them is similar to accessing the instance variable:
  - `tally.incrementValue( );`
  - `int current = tally.getValue( );`

Encapsulation

- Objects should be self-governing.
- They should control their own instance variables.
- You implement this by making the instance variables `private`.
- This is called *hiding* the instance variables.
  - Variable hiding prevents direct variable access by external clients.
- Hiding the instance variables makes the object self-contained.
  - It's as if the object's instance variables are in a capsule.
  - This is why instance variable hiding is usually called *encapsulation*.

Why is Encapulation Needed?

- Direct attribute access is unsafe/dangerous.
  - A malicious external agent may corrupt the attribute's value.
  - A typo may result in an undetected (semantic) error.
- Encapsulation simplifies the complexity of the API.
  - There are fewer possible scenarios.
  - Makes desiging the API easier.
  - Makes reasoning about the API easier.
  - Makes testing easier.
- Prevents clients from *depending* on the implementation.
  - Allows implementation changes without breaking clients.

Contract

- We hide all instance variables.
- We hide all methods shouldn't be part of the API

Hiding Methods

- Java also lets you hide method declarations.

Java

```
public int squareOfAnswer( ) {
    return answer( ) * answer( );
}

private int answer( ) {
    return 42;
}
```

- Hiding methods has similar advantages as hiding attributes.

For Monday

- Read Section 1.6.
- Read Section 2.1–2.5.

Acknowledgements

- This lecture corresponds to [*Big Java, Early Objects*, 2.1–2.6].

Bibliography


Horstfmann, Cay S. *Big Java, Early Objects.* International Student Version. Wiley. ISBN: 978-1-118-31877-5.

About this Document

- This document was created with `pdflatex`.
- The LaTeX document class is `beamer`.