Understanding class definitions
Lecture 3

Waterford Institute of Technology

January 27, 2015

John Fitzgerald

Documenting a program

Good documentation of program code

- Helps code maintenance
- Adds to value of project

```
/**
* Fixed-price machine
*/
public TicketMachine(int ticketCost)
{
    //price of tickets in cents
    price = ticketCost;
}
```

Note two methods of commenting

- // *inline:* Comment here
- /* *block:* Comment here */

Style guide

Style guide provides guidelines on layout, indentation, capitalization

- Order of class parts
  - Fields
  - Constructors
  - Methods
- Upper case class name
- Lower case field & method names
  - `Student student`
  - //Example camel case
  - `String getName()`

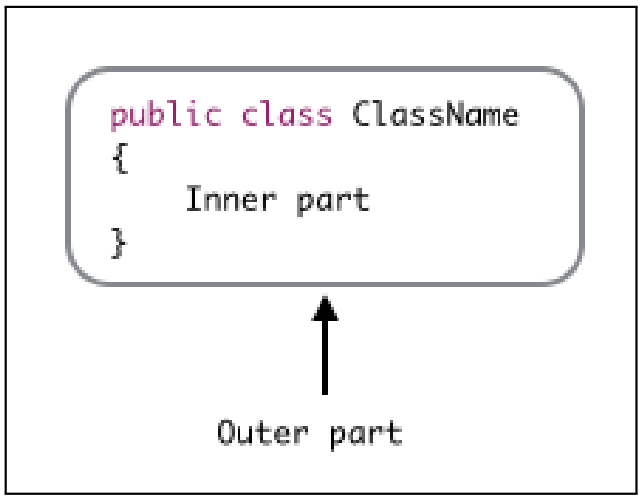
```
public class Circle
{
    private int diameter;
    private int xPosition;
    private int yPosition;
    private String color;
    private boolean isVisible;

    /**
     * Create a new circle at
     */
    public Circle()
    {
        diameter = 30;
        xPosition = 20;
        yPosition = 60;
        color = "blue";
        isVisible = false;
    }
```

Class components

Class decomposable into

- Outer wrapping or class header
- Inner part containing
  - Constructor
  - Fields
  - Methods

Convention: class name begins with upper case



Class components

Class comprises

- Fields: store data
- Constructors: builds object at instantiation
- Methods: provide object functionality

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

Order of class parts

Parts order matter of style

Our style order

1. Fields
2. Constructors
3. Methods

```
public class TicketMachine
{
    private int price;
    public TicketMachine() { ... }
    public String getPrice() { ... }
}
```

Class layout

Example class layout

- Outer part (class header) only
- This a valid class definition
- Compiles and executes
- Does nothing

```
/**
 * This class compiles ok despite
 * having no fields, explicit constructor nor methods
 */
public class TicketMachine
{
        //TODO: Inner part here
}
```

Reserved words

Java 7 has list 50 reserved words
Also known as keywords

- Only allowed to use for designated role
- Example keyword: *private*
- Allowed: `private String privateSoldier;`
- Disallowed: `private String private;`

Class and instance variables

Class variable

- Preceeded by static
- Same value all class objects
- static int counter;

Instance variable

- Attributes differ across objects
- Example: int counter;

```
public class TicketMachine
{
        static private int counter = 0;
        private int price;
        private String name;
}
```

Naming variables

Rules for naming variables

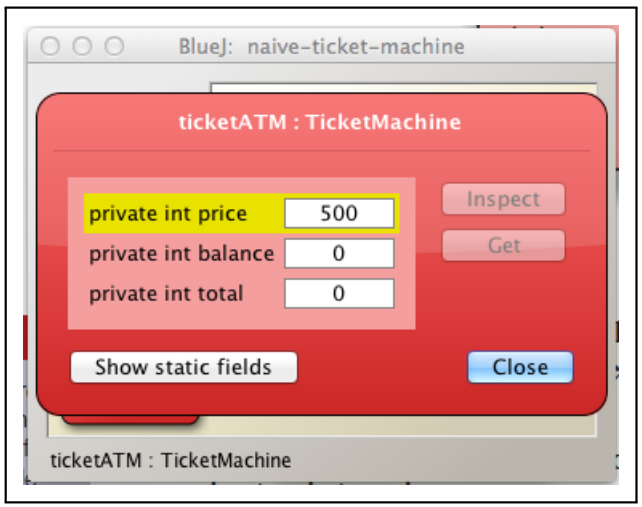
- Any legal identifier permitted
- Unlimited sequence of Unicode characters
  - Legal to begin with letter, dollar($) or underscore (_)
  - Convention:: begin with lower case letter
- Case-sensitive : these different
  - `int treebase`
  - `int treeBase`
- Choose self-documentating, non-cryptic names
  - Good: `int speed`
  - Bad: `int s`
  -
- Variable consisting of more than one word
  - Use camel case: `int treeOakBase`
  - Rather than: `int tree_oak_base`

Fields

Fields reserve space within object
Data stored in this space
Data sometimes referred to as *attribute* When object created fields may have values

- Assigned during creation
- Assigned later
- Changed later

Because fields modifiable also called *variables*



Assignment

Values stored in field variables can be modified.

- `balance` = 500
- Original value in *balance* replaced by 500
- = is assignment operator

Assignment Operators

Table 1 : Frequently used assignment operators

$+=$	`a +=` 10;	adds 10 to a
$-=$	`a -=` 10;	subtracts 10 from a
$*=$	`a *=` 10;	multiplies a by 10
$/=$	`a /=` 10;	divides a by 10

Unary Operators

Table 2 : Increment and Decrement Operators

++	`a++;`	adds 1 to a
−−	`a−−;`	subtracts 1 from a

Equality and Relational Operators

Table 3 : Equality and Relational Operators

Equal to	==
Not equal to	!=
Greater than	>
Greater than or equal to	>=
Less than	<
Less than or equal to	<=

Access control

Access level modifiers precede fields, methods and constructors.

- private: visible only within class
- public: visible to world

```
//This field visible only within own class
private int price;
//These methods acessible to objects not of own class(es)
public void setPrice(int price);
public String getPrice();
```

Constructor

Constructor engaged in object creation (instantiation)
Construction process called *initialization*
Constructors :

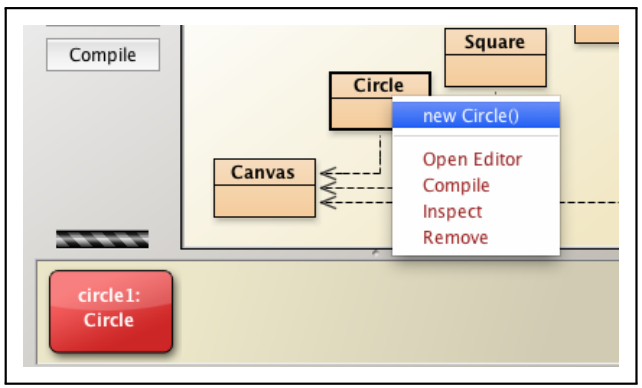
- Have same name as class
- May have zero, one or more arguments (parameters)
- A class may have more than one constructor
- If constructor not included one provided transparently by compiler
- Do not return values

Constructor continued

Have already invoked constructor
Example: *shapes* project when creating circle object

- new `Circle()`

This expression creates new Circle object by invoking its constructor
Note absence parameters this instance



References

1. The Java Language Specification

`http://docs.oracle.com/javase/specs/jls/se7/html/`
[Accessed 2014-02-25]

2. Operators

`http://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html`

[Accessed 2014-02-25]

3. Summary of Operators

`http://docs.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html`

[Accessed 2014-02-25]