

# Understanding class definitions

## Lecture 4

Waterford Institute of Technology

January 19, 2016

John Fitzgerald

# Presentation outline

Estimated duration presentation

Questions at end presentation - helps to note slide numbers

Topics discussed:

- Parameters - formal & actual
- Scope - visibility & lifetime variables
- Styles
- Expression, statement, block
- Getters & setters (Accessors & Mutators)
- Print statement
- Conditional statement - facilitates branching
- JavaDoc tool

# Parameter data

Constructors & methods receive data from parameters

Parameters also known as arguments

Defined in constructor or method header

- `public TicketMachine(int ticketCost)`
- `public void insertMoney(int amount)`

Data assigned to object field *price*

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

**ticketCost** referred to as *formal parameter*

Value passed, e.g. 500 *actual parameter*

# Parameter data

## Formal and actual parameters

*int amount* is the formal parameter

- `public void makeDeposit(int amount)`

*100* below example actual parameter (argument)

```
public void makeDeposit(int deposit)
{
    this.balance += deposit;
}
.
.
bank.makeDeposit(100);
```

# Scope

Scope of variable is block of code within which variable visible  
Consider this code:

```
public class TicketMachine
{
    int price;
    public TicketMachine(int ticketCost)
    {
        price = ticketCost;
    }
    ...
}
```

Field **price** visible

- Throughout entire class
- But **ticketCost** visible only in constructor

# Local variables

Consider this code snippet:

```
public int refundBalance()  
{  
    int refundAmount;  
    refundAmount = balance - adminCost;  
    return refundAmount;  
}
```

**refundAmount** a local variable, not a field

- Defined inside method
- `int refundAmount;` is *declaration*
- Simultaneous declaration and initialization valid
- `int refundAmount = balance - adminCost;`
- `return balance-adminCost;` //avoid local variable

# Verbose v concise styles

## Verbose

```
public int refundBalance()  
{  
    int refundAmount;  
    refundAmount = balance - adminCost;  
    return refundAmount;  
}
```

## Concise

```
public int refundBalance()  
{  
    return balance - adminCost;  
}
```

# Source code format

Different styles of layout format used  
positioning curly brace pairs `{ }`

Recommendation: use one style only throughout project

Styles mixed throughout presentation for layout convenience

```
public void insertMoney(int amount)
{
    if (amount > 0) {
        balance = balance + amount;
    }
}
```

```
public void insertMoney(int amount)
{
    if (amount > 0)
    {
        balance = balance + amount;
    }
}
```



# Lifetime of variable

## Variable lifetime

- *ticketCost* invisible outside constructor
- But field *price* and object lifespans same
- *price* dies when object destroyed

```
public class TicketMachine
{
    int price;
    public TicketMachine(int ticketCost)
    {
        price = ticketCost;
    }
}
```

# Revisit class structure

- Fields: instance or class variables.
- Constructors have same name as class
- Class has same name as file
- Method naming at discretion of developer
- Invoke (call) constructor using **new** operator
- Invoke method directly within object
  - `getRadius();`
- Invoke method on object
  - `circle.getRadius();`

```
public class Circle
{
    double diameter;
    public Circle() {
        setDiameter(50);
    }
    void setDiameter(double diameter) {
        this.diameter = diameter;
    }
}
```

```
public class Main
{
    static Circle circle = new Circle();
    void test() {
        circle.setDiameter(100);
    }
}
```

# Expressions, statements, blocks

## Expression

- Comprises variables, operators & methods
- Constructed to evaluate to single value
- The following expression evaluates to 3
- $1 + 2$ ;

## Statement

- Like sentences in language
- Line of code that performs some action
- `int savings = 1.1*(savings + getBonus());`

**Block:** Group statements within corresponding pair braces

- ```
public void aMethod(int val)
{
    value = val;
}
```

# Accessors

## Accessor (Getter)

- Value of field unchanged
- `getNumber`
- `getStudent`

```
public class SomeClass
{
    private int number;
    private Student student;

    public int getNumber()
    {
        return number;
    }
    public Student getStudent()
    {
        return student;
    }
    ...
}
```

# Mutators

Mutator (Setter)  
changes field value

- updateBalance
- setCustomer

```
public class BIABank
{
    private int balance;
    private Customer customer;

    public void updateBalance(int deposit)
    {
        balance += deposit;
    }
    public void setCustomer(Customer customer)
    {
        this.customer = customer;
    }
}
```

# Printing

```
/**
 * Print a ticket.
 */
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();
}
```

```
#####
# The BlueJ Line
# Ticket
# 500 cents.
#####
```

# Printing continued

printTicket method analysed:

- `void printTicket():` Method signature
- **System:** An inbuilt Java class
- **out:** *Printstream out* is System field
- **println:** is Printstream method
- `"# " + price + " cents."`

concatenates to single string

# Conditional statements

Facilitates choice

Accept only positive sum

- If amount greater than zero
  - Update balance
- Else amount zero or less
  - Print error message

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



# Conditional statements

## Multiple branching options

```
class IfElseDemo {  
    public static void main(String[] args) {  
  
        int testscore = 76;  
        String grade;  
  
        if (testscore >= 90) {  
            grade = "A";  
        } else if (testscore >= 80) {  
            grade = "B";  
        } else if (testscore >= 70) {  
            grade = "C";  
        } else if (testscore >= 60) {  
            grade = "D";  
        } else {  
            grade = "F";  
        }  
        System.out.println("Grade = " + grade);  
    }  
}
```

# Class documentation

## shapes project documentation

**All Classes**  
[Canvas](#)  
[Circle](#)  
[RectangleBJ](#)  
[Square](#)  
[Triangle](#)

**Package** **Class** **Tree** **Index** **Help**  
[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)  
SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL: FIELD](#) | [CONSTR](#) | [METHOD](#)

### Class Circle

[java.lang.Object](#)  
└ **Circle**

```
public class Circle
extends Object
```

A circle that can be manipulated and that draws itself on a canvas.

**Version:**  
2006.03.30

**Author:**  
Michael Kolling and David J. Barnes

### Constructor Summary

|                           |                                                             |
|---------------------------|-------------------------------------------------------------|
| <a href="#">Circle</a> () | Create a new circle at default position with default color. |
|---------------------------|-------------------------------------------------------------|

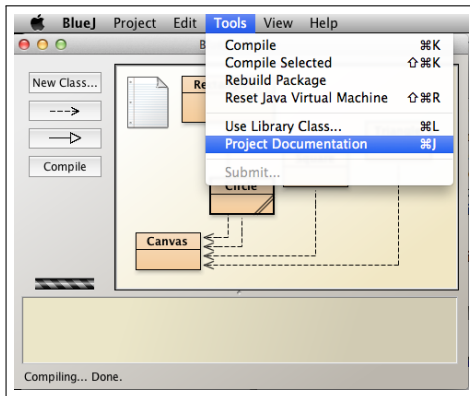
### Method Summary

|      |                                                                |                                              |
|------|----------------------------------------------------------------|----------------------------------------------|
| void | <a href="#">changeColor</a> ( <a href="#">String</a> newColor) | Change the color.                            |
| void | <a href="#">changeSize</a> (int newDiameter)                   | Change the size to the new size (in pixels). |
| void | <a href="#">makeInvisible</a> ()                               | Make this circle invisible.                  |

# Class documentation

Generate *javadoc*

- BlueJ Menu
  - Tools
  - Project Documentation



# Class documentation

Different philosophies exist on how to document

- Good documentation adds value to application
- Choose descriptive class, variable and method names
  - Bad : `int r = 10;`
  - Good: `int radius = 10;`
- Avoid naive comments such as
  - `radius = 10; //sets radius = 10`
- State the reason for code, the *why*
- Write code so that *what* code does is self-evident
- Make commenting code the norm

# Class documentation

## Comment block

```
/**
```

```
 * this is a comment
```

```
 */
```

```
/**
```

```
 * Changes Color color attribute value and redraws object
```

```
 * @param newColor this color used in rendering all objects
```

```
 */
```

```
public void changeColor(String newColor)
```

```
{
```

```
    color = newColor;
```

```
    draw();
```

```
}
```

Question: do you think the above is a good comment?

# Class documentation

## Use approved tags

- @param
- @return

```
/**  
 * Accessor to retrieve color  
 * @return the color of the rectangle  
 */  
public String getColor()  
{  
    return color;  
}
```

# TicketMachine JavaDoc output extract

```
/**
 * Create a machine that issues tickets of the given price.
 * Note that the price must be greater than zero.
 *
 * @param ticketCost is the price of the current ticket.
 */
public TicketMachine(int ticketCost)
{
    if (ticketCost > 0)
    {
```

## Constructor Detail

### TicketMachine

```
public TicketMachine(int ticketCost)
```

Create a machine that issues tickets of the given price. Note that the price must be greater than zero.

#### Parameters:

ticketCost - is the price of the current ticket.

# Summary

## Understanding class definitions

- Comment: documentation only, not executed
- Constructor: sets up object at creation (instantiation)
- Scope: defines section in code where variable visible
- Lifetime: how long variable exists before destruction
- Accessor: returns information on state of object
- Mutator: changes object state
- Conditional: takes one of two actions following test
- Local variable: declared within single method with same scope & lifetime as method



## Referenced material

1. How to Write Doc Comments for the Javadoc Tool

[www.oracle.com/technetwork/java/javase/documentation/index-137868.html](http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html)

[Accessed 2014-02-09]

2. Barnes D.J. Kolling M. Objects First with Java. Third Edition. Pearson Education Ltd. 2003

3. Expressions, statements, blocks

<http://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html>

[Accessed 2014-02-09]

## Referenced material

### 4. System API

<https://docs.oracle.com/javase/7/docs/api/java/lang/System.html>

[Accessed 2014-02-09]

### 5. Javadoc documentation generator

<https://en.wikipedia.org/wiki/Javadoc>

[Accessed 2016-01-13]