
CSE 142

Class Implementation in Java

1/10/2003

(c) 2001-3, University of Washington

F-1

Outline

- Implementing classes in Java
- Instance variables – properties
- Value-returning methods for queries
- Void methods for commands
- Return statement
- Assignment statement and arithmetic expressions
- Method parameters
- Constructors

1/10/2003

(c) 2001-3, University of Washington

F-2

Specification vs Implementation - Review

- Specification – external view of an object/class
 - View of the class as seen by *client* code (i.e., other code that creates or uses instances – objects – of this class)
 - Class name and method names, parameters, and descriptions
- Implementation – internal details private to the class
 - Instance variables – properties
 - Statements that describe algorithms carried out by methods

1/10/2003

(c) 2001-3, University of Washington

F-3

Instance Variables

- Example in class `BankAccount`

```
private int number;           // account number
private String name;         // account name
private double balance;      // current balance
```

- These are instance ***variable declarations***

```
private <type> <identifier>
```

- `private` – part of the implementation, not visible outside
- `<type>` - the type of the variable
- `<identifier>` - a (hopefully meaningful) name for the variable
- Each object of class `BankAccount` will have its own set of instance variables

1/10/2003

(c) 2001-3, University of Washington

F-4

Implementing Methods for Simple Queries

- **Example in class BankAccount**

```
/** return the current balance of this BankAccount */
public double getBalance() {
    return balance;
}
```

- **When this method is executed, it replies with the value of the instance variable balance**

```
checking.getBalance()
```

1/10/2003

(c) 2001-3, University of Washington

F-5

More About Value-Returning (Query) Methods

- **Form**

```
/** Comment specifying the method */
public <result type> <identifier> () {
    list of statements
}
```

- **Details**

- **public** – this method is part of the public specification of the class (methods can also be private; we'll see examples eventually)
- **<result type>** – the type of the value returned by this query
- **<identifier>** – the (hopefully meaningful) name of this method
This is the name of the query that the method implements
- **list of statements** – the body of the method
These make up the algorithm that the method executes when it is called

1/10/2003

(c) 2001-3, University of Washington

F-6

Return Statement

- **First example of a statement**

```
return expression;
```

- **Meaning**

- **Evaluate the expression to get a value**

In `getBalance`, the expression is the name of the instance variable `balance`
For a variable, evaluation means get its current value

- **Then, finish execution of this method (query), replying with the value of the expression**

1/10/2003

(c) 2001-3, University of Washington

F-7

println vs *return*

```
public int tryPrintlnAndReturn() {
    System.out.println(1);
    return 2;
}
```

- **What does this method print?**
- **What does this method return?**
- **Hint: Don't go by how it looks in the Interactions pane of Dr Java! What do you get if you type each of these:**

```
tryPrintlnAndReturn()
tryPrintlnAndReturn();
```

1/10/2003

(c) 2001-3, University of Washington

F-8

println, return Puzzlers: Which are Errors?

```
public void tryPrintlnAndReturn() { //A
    System.out.println(1);
    return 2;
}
public int tryPrintlnAndReturn() { //B
    return 2;
    System.out.println(1);
}
public void tryPrintlnAndReturn() { //C
    System.out.println("1");
}
public int tryPrintlnAndReturn() { //D
    System.out.println("1");
    return "2";
}
```

1/10/2003

(c) 2001-3, University of Washington

F-9

Arithmetic Expressions

- **Basic components**
 - Literals – 17, 3.0, 1.023e23
 - Variable names – value is the current value of the variable
- **Operators (see book for all the details)**
 - +, -, *, /, % (remainder)
 - Gotchas: for ints, x/y yields integer part, dropping any fraction: x%y gives the remainder
 - Operators have the usual precedence
 - For example, a + b * c is understood to mean a + (b * c)
 - Binary operators (ones that have two components) are left associative: a * b / c means (a * b) / c
 - Use parentheses where needed to override: a * (b / c)
 - **Mixing ints and doubles is normally ok – the int is converted to a double and the calculation is done as a double**

1/10/2003

(c) 2001-3, University of Washington

F-10

Exercise – Another Query

- **Complete the query in class BankAccount**

```
/** return the name of this BankAccount */
public double getName() {
```

```
}
```

1/10/2003

(c) 2001-3, University of Washington

F-11

Implementing Methods for Simple Commands

- **Example in class BankAccount**

```
/** Set this BankAccount's name to newName */
public void setName(String newName) {
    name = newName;
}
```

- **When this method is executed, it changes the name instance variable; it does not return a value**
 - Executed only for its effect

1/10/2003

(c) 2001-3, University of Washington

F-12

More About Command Methods

• Form

```
/** Comment specifying the method */
public void <identifier> (parameters) {
    list of statements
}
```

• Details

- **public**, **<identifier>**, and **list of statements** – same as for queries
- **void** – Indicates that this is a command that doesn't return a value (as opposed to the result type of a query)
- **parameters** – information supplied with command message
Same form as a variable declaration
(Note: Queries can also have parameters, but they have not been needed in the simple cases we've seen so far)

1/10/2003

(c) 2001-3, University of Washington

F-13

Assignment Statement

• Second example of a statement

```
variable = expression ;
```

• Meaning

- First, evaluate the expression to get a value
- Second, bind that value to the *variable* whose name appears on the left
- These two steps are done in that order, not simultaneously
- Question: what does this mean (or do)?

```
count = count + 1;
```

1/10/2003

(c) 2001-3, University of Washington

F-14

Exercise – Another Simple Command

• Complete the command in class BankAccount

```
/** Set this BankAccount's number to newNumber */
public void setNumber(int newNumber) {
```

```

}
```

1/10/2003

(c) 2001-3, University of Washington

F-15

Constructor

• Example in class BankAccount

```
/** Construct a new BankAccount with balance=number=0 and no name */
public BankAccount() {
    number = 0;
    name = "";
    balance = 0.0;
}
```

• This is a lot like a command method. Difference – it is executed automatically each time a new BankAccount instance is created

- Idea: Use the constructor to initialize newly created objects to some sensible state
- Syntax difference from other methods: no result type or void

1/10/2003

(c) 2001-3, University of Washington

F-16

Creating and Using BankAccount Objects

- Before going further, we'd better test what we've done

```
BankAccount savings = new BankAccount();
savings.setName("A. Hacker");
savings.setNumber(4200);
savings.getName();
```

1/10/2003

(c) 2001-3, University of Washington

F-17

A Smarter Constructor

- Better would be to provide initial values for name, account number, and balance when we create a BankAccount

- Solution: use parameters in the constructor

```
/** Construct a new BankAccount with given account name, number, and balance */
public BankAccount(String accountName, int accountNumber, double initialBalance) {
    number = accountNumber;
    name = accountName;
    balance = initialBalance;
}
```

- Test

```
BankAccount checking = new BankAccount("E. Fudd", 4179, 42.17);
checking.getName();
checking.getBalance();
```

1/10/2003

(c) 2001-3, University of Washington

F-18

Deposit – Another Command

- In class BankAccount

```
/** Deposit given amount in this BankAccount */
public void deposit(double amount) {
    balance = balance + amount;
}
```

- Meaning is clear since expression in assignment statement is evaluated before balance is changed

1/10/2003

(c) 2001-3, University of Washington

F-19

Transfer – Objects as Parameters

- From class BankAccount

```
/** Transfer the given amount from otherAccount to this BankAccount */
public void transfer(double amount, BankAccount otherAccount) {
    balance = balance + amount;
    otherAccount.balance = otherAccount.balance - amount;
}
```

- Instance variable (field) access

```
objectName.variableName
```

is a reference to the given instance variable of the given object

- Legal in the example because otherAccount is another instance of BankAccount. Since transfer is part of class BankAccount, it can access private information in any BankAccount

1/10/2003

(c) 2001-3, University of Washington

F-20

Summary

- **Implementation of classes**
 - Instance variables – type plus name
 - Methods – statements that make up the body of each method
- **Statements**
 - return
 - Assignment & arithmetic expressions
- **Creating objects and calling methods**

- **Coming attractions**
 - More details about objects, method calls, and variables
 - More complex statements – conditionals and loops