

1B1a

Using Methods

1 Copyright © 2003, Graham Roberts Department of Computer Science

Agenda

- Writing programs using methods.
- Good method design.
- Methods, objects and classes.

2 Copyright © 2003, Graham Roberts Department of Computer Science

Using Methods

- Instead of writing a long list of statements and executing from start to finish...
- We can now write a collection of methods that call each other.

3 Copyright © 2003, Graham Roberts Department of Computer Science

Calling methods

```
graph TD; main --> M1; main --> M4; M1 --> M2; M1 --> M3; M4 --> M5;
```

- main calls M1
- M1 calls M2 and M3
- main calls M4
- M4 calls M5

Methods can belong to different object but could belong to the same object.

4 Copyright © 2003, Graham Roberts Department of Computer Science

Programming Strategy

- Prior to OOP, languages only had procedures/functions.
- This led to a style of programming known by various names:
 - Procedural decomposition
 - Top-down programming
 - Structured programming
 - and others

5 Copyright © 2003, Graham Roberts Department of Computer Science

An approach to problem solving

- Decompose a problem into a sequence of simpler methods and then call each method.
- For each simpler method repeat the process.
- Stop when a method becomes trivial to write.
 - E.g., doing payroll processing.

6 Copyright © 2003, Graham Roberts Department of Computer Science

Methods and OO Programming?

- The same basic decomposition principle applies to OO programs,
- BUT we also have to decide which objects provide which methods.
 - Allocation of behaviour to objects.
 - (Actually it's not that simple...)

7

Copyright © 2003, Graham Roberts

Department of Computer Science



Choosing methods

- A method should focus on doing one thing well:
 - Compute and return a value.
 - Perform an action.
- Be *cohesive*.
- If a method becomes more complicated and tries to do several things:
 - Split it up into smaller methods!

8

Copyright © 2003, Graham Roberts

Department of Computer Science



Method size

- Keep methods small:
 - “Small enough but no smaller.”
- If a method is longer than ~15 lines then always consider splitting it.
- Don't go mad and reduce all methods to one line though!
- Develop your feel for what makes a good method.

9

Copyright © 2003, Graham Roberts

Department of Computer Science



Method parameters

- Use parameters to make a method more general purpose.
 - Write $f(n)$, rather than $f1()$, $f2()$, $f3()$, etc.
- Typically, few methods need more than 3 parameters.

10

Copyright © 2003, Graham Roberts

Department of Computer Science



Questions?

11

Copyright © 2003, Graham Roberts

Department of Computer Science



A Simple One Class Example

- Program specification:
 - Read in a sequence of integers, sort them, and output the sorted sequence.
- The problem can be simply decomposed into three methods: read, sort and output.

12

Copyright © 2003, Graham Roberts

Department of Computer Science



The main method

All the methods will be declared in this class.

```
public class ReadNSort
{
    public static void main(String[] args)
    {
        ReadNSort myObj = new ReadNSort();
        myObj.readIntegers();
        myObj.sortIntegers();
        myObj.outputIntegers();
    }
}
```

13 Copyright © 2003, Graham Roberts

Department of Computer Science



readIntegers

- Use a loop to read in 100 integers.
- What do we do with the integers?
 - Choose an appropriate data structure.
- Let's put them in an array.

14 Copyright © 2003, Graham Roberts

Department of Computer Science



readIntegers take 2

```
public void readIntegers()
{
    int[] myInts = new int[100];
    // ... Loop to read in integers
}
```

- Wait... What happens to the array?

15 Copyright © 2003, Graham Roberts

Department of Computer Science



readIntegers take 3

```
public int[] readIntegers()
{
    int[] myInts = new int[100];
    // ... Loop to read in integers
    return myInts;
}
```

- Change return type and return array.

16 Copyright © 2003, Graham Roberts

Department of Computer Science



sortIntegers

- Now needs to take and return an array argument:

```
public int[] sortIntegers(int[] myInts)
{
    // Do the sorting
    return myInts;
}
```

17 Copyright © 2003, Graham Roberts

Department of Computer Science



outputIntegers

- Only needs to have an array argument:

```
public void outputIntegers(int[] myInts)
{
    // loop to output integers
}
```

18 Copyright © 2003, Graham Roberts

Department of Computer Science



Amend main method:

```
public static void main(String[] args)
{
    ReadNSort myObj = new ReadNSort ();
    int[] myInts = myObj.readIntegers();
    myInts = myObj.sortIntegers(myInts);
    myObj.outputIntegers(myInts);
}
```

19 Copyright © 2003, Graham Roberts

Department of Computer Science



A Variant

```
public static void main(String[] args)
{
    ReadNSort myObj = new ReadNSort ();
    int[] myInts = new int[100];
    myObj.readIntegers(myInts);
    myObj.sortIntegers(myInts);
    myObj.outputIntegers(myInts);
}
```

- Is this way better?

20 Copyright © 2003, Graham Roberts

Department of Computer Science



Probably...

- readInteger now only does *one* thing (read in 100 integers).
- Creating the array makes it a little less general purpose, less cohesive.
- Of course, the array size is fixed, so the next variant should change that (an exercise for you).

21 Copyright © 2003, Graham Roberts

Department of Computer Science



Questions?

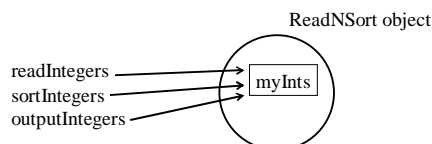
22 Copyright © 2003, Graham Roberts

Department of Computer Science



Another solution...

- Our read, sort, output program could be a little more object-oriented...
 - The object should store the array.



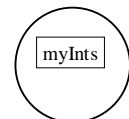
23 Copyright © 2003, Graham Roberts

Department of Computer Science



Instance Variable

- An *instance variable* belongs to an object.
- Can be used by any *instance methods*.
- Doesn't need to be passed as a parameter,
 - Or declared in main.



24 Copyright © 2003, Graham Roberts

Department of Computer Science



Declaring instance variable

```
public class ReadNSort
{
    private int[] myInts = new int[100];
    // Methods ...

    public static void main(String[] args)
    {
        ReadNSort myObj = new ReadNSort ();
        myObj.readIntegers();
        myObj.sortIntegers();
        myObj.outputIntegers();
    }
}
```

25 Copyright © 2003, Graham Roberts

Department of Computer Science



Private ...

- `private int[] myInts = new int[100];`
- The instance variable is created with the object (by new expression).
- Is *private* to the object
 - Only accessible to the instance methods
 - Not to main
- Declared in *class scope*.
- Lifetime is the same as the object.

26 Copyright © 2003, Graham Roberts

Department of Computer Science



Simpler methods

```
public void readIntegers()
{
    // ... Loop to read in integers
}
```

- Array `myInts` is in scope and accessible to method.
- No parameter, no return statement, no array creation, void method.

27 Copyright © 2003, Graham Roberts

Department of Computer Science



Simpler methods (2)...

```
public void sortIntegers()
{
    // Do the sorting
}
public void outputIntegers()
{
    // loop to output integers
}
```

28 Copyright © 2003, Graham Roberts

Department of Computer Science



Responsibilities

- The *object* is responsible for holding the array.
- The object provides the services of reading, sorting and displaying data.

29 Copyright © 2003, Graham Roberts

Department of Computer Science



Questions?

30 Copyright © 2003, Graham Roberts

Department of Computer Science



But...

- In some other class:

```
public void f()
{
    ReadNSort obj = new ReadNSort();
    obj.outputIntegers();
    obj.sortIntegers();
    obj.readIntegers(); //Whoops.....
}
```

31 Copyright © 2003, Graham Roberts

Department of Computer Science



Contract

- The object supports its responsibilities,
- But also needs a contract with the *client* so that its responsibilities are used correctly.
 - Use the object according to its specification.

32 Copyright © 2003, Graham Roberts

Department of Computer Science



Specifying a Contract

- The order of method call should be specified in the class declaration.
 - Comments, javadoc.
- Unfortunately, can't be enforced via language syntax.
- Design and programming issue.
 - Programmer must get it right!

33 Copyright © 2003, Graham Roberts

Department of Computer Science



Specifying a Contract (2)

- Can include code to check and report errors at runtime:

```
boolean inputDone = false;
...
public void sortIntegers()
{
    if (!inputDone)
    {
        System.out.println("no input done");
        return;
    }
}
```

But messy and needs lots of checking code.

34 Copyright © 2003, Graham Roberts

Department of Computer Science



Rethink interface

- Why are read, sort and display methods public?
- Make them private!
- Provide a single public method to control object...

35 Copyright © 2003, Graham Roberts

Department of Computer Science



Public/Private

```
private void readIntegers() { ... } // Can only be used by
private void sortIntegers() { ... } // other methods in
private void outputIntegers() { ... } // same class.
```

```
public void readSortOutput() // new control method
{
    readIntegers(); // Enforce correct order.
    sortIntegers();
    outputIntegers();
}
```

36 Copyright © 2003, Graham Roberts

Department of Computer Science



The client

- Can now only call `readSortOutput`.

```
public static void main(String[] args)
{
    ReadNSort obj = new ReadNSort();
    obj.readSortOutput();
    // obj.readIntegers(); // Error, method is private
}
```

37

Copyright © 2003, Graham Roberts

Department of Computer Science



Public v. Private

- Collection of small, cohesive methods for private implementation.
- Minimal public interface
 - Control method public, supporting methods private.
 - Thin interface (as opposed to fat interface).

38

Copyright © 2003, Graham Roberts

Department of Computer Science



So,

- Use methods to decompose code into small, cohesive units.
- But make minimum number of methods public.
 - Provide control and/or access methods.

39

Copyright © 2003, Graham Roberts

Department of Computer Science



Insides v. Outsides

- Distinguish structure needed for implementation of object,
- From structure made public to users (clients) of object.
- Think Abstraction.

40

Copyright © 2003, Graham Roberts

Department of Computer Science



Parameters v. instance variables?

- In this 1-class example parameters were eliminated.
 - Nature of the problem.
- Parameters still essential:
 - Function-like methods.
 - Generic methods.
 - Passing information from one object to another.

41

Copyright © 2003, Graham Roberts

Department of Computer Science



Questions?

42

Copyright © 2003, Graham Roberts

Department of Computer Science



Top-down programming?

- Long history, still widely used for non-OO languages.
- It is still relevant?
- Do we want to design and write programs in this way?

Yes and No!

43 Copyright © 2003, Graham Roberts

Department of Computer Science



Yes...

- Top-down decomposition is useful for solving small-scale problems,
 - Private methods providing services to an object's public methods.
- and, in some cases, for solving sub-parts of a larger problem.

44 Copyright © 2003, Graham Roberts

Department of Computer Science



No...

- Top-down decomposition is a poor way for solving larger problems and for designing larger programs.
- In fact, it's often a **complete disaster!**
- The approach simply does not scale-up.

45 Copyright © 2003, Graham Roberts

Department of Computer Science



A better way

- Object-oriented design and programming provides much better solutions to the construction of larger programs.
- As the year continues we will examine why in detail.
- But also read the text book now.

46 Copyright © 2003, Graham Roberts

Department of Computer Science



For now...

- Let's consider route planning...
- Top-down solution is rigid, inflexible and fragile.
- OO solution is general purpose, flexible and robust.

47 Copyright © 2003, Graham Roberts

Department of Computer Science



Summary

- Programs can be decomposed into a collection of methods that call one another.
- Methods should be small and cohesive.
- Method based decomposition is suitable for solving small scale problems or sub-problems only.
- But methods are an essential building block of object-oriented programs.

48 Copyright © 2003, Graham Roberts

Department of Computer Science

