

1B1b

Classes in Java

Part III

1 Copyright © 2004, Graham Roberts

Department of Computer Science



Review

- We have seen that classes:
 - declare instance variables.
 - declare methods.
 - may have constructors.

Now want to start filling in further details.

2 Copyright © 2004, Graham Roberts

Department of Computer Science



Static

Why are some methods and variables declared as static?

3 Copyright © 2004, Graham Roberts

Department of Computer Science



Static (2)

It depends on whether variables or methods “belong” to the class or to instance objects of the class.

4 Copyright © 2004, Graham Roberts

Department of Computer Science



Static (3)

- Non-static variables are *instance variables*.
 - Each object gets its own copy of each variable.
- Static variables are *class variables*.
 - A *single* copy of each variable exists and can be accessed by any other method in the class.

```
class Test
{
    private int instanceVar;
    private static int classVar;
}
```

5 Copyright © 2004, Graham Roberts

Department of Computer Science



Example

- Count number of times a method is called for *all* instance objects of a class.

```
private static int count = 0;
public void f()
{
    count++;
    // Rest of method...
}
```

6 Copyright © 2004, Graham Roberts

Department of Computer Science



final

- Public static variables are often used to create symbolic constants.
 - E.g., Math.PI
- Such variables are additionally declared *final*:
 - public static final double PI = 3.141...
- The value of a final variable cannot be changed by assignment.

7 Copyright © 2004, Graham Roberts

Department of Computer Science



Static (4)

- Non-static methods are *instance methods*.
 - An instance method must be called for an object of the class.
 - x.method(args);
- Static methods are *class methods*.
 - A static method can be called by any method declared by the class, or any method at all if public.

8 Copyright © 2004, Graham Roberts

Department of Computer Science



Example – Singleton

```
private static MyClass instance;
private MyClass() { ... }
public static MyClass getInstance()
{
    if (instance == null)
        instance = new MyClass();
    return instance;
}
```

9 Copyright © 2004, Graham Roberts

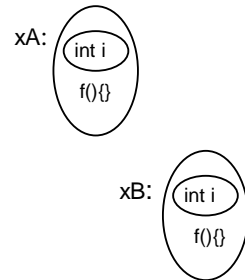
Department of Computer Science



Static (5)

```
public class X
{
    private int i;
    public void f(){}
}

X xA = new X();
X xB = new X();
xA.f();
xB.f();
```



10 Copyright © 2004, Graham Roberts

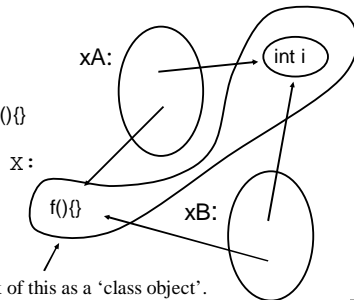
Department of Computer Science



Static (6)

```
public class X
{
    private static int i;
    public static void f(){}
}

X xA = new X();
X xB = new X();
xA.f();
xB.f();
X.f();
```



11 Copyright © 2004, Graham Roberts

Department of Computer Science



Questions

12 Copyright © 2004, Graham Roberts

Department of Computer Science



Classes and Program Structure

- A program consists of a collection of classes.
- Those classes define the abstract structure of the program in terms of the relationships between the classes.
- When the program is run, the relationships are realised by object references.

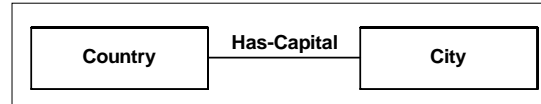
13 Copyright © 2004, Graham Roberts

Department of Computer Science



Representing Associations

- An association between two classes typically means that an object of one class has a reference to an object of another class.



14 Copyright © 2004, Graham Roberts

Department of Computer Science



Association (2)

```

class Country
{
    private City capital;
    ...
}
    
```

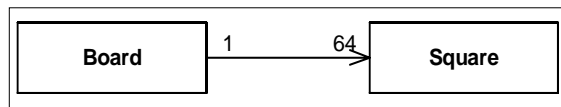
15 Copyright © 2004, Graham Roberts

Department of Computer Science



Association (3)

- The type used to represent the association needs to be determined correctly. For example:



16 Copyright © 2004, Graham Roberts

Department of Computer Science



Association (4)

```

class Board
{
    private Square[] squares
    = new Square[64];
    // or
    // ArrayList squares = new ArrayList();
    ...
}
    
```

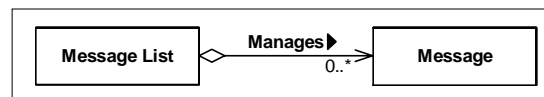
17 Copyright © 2004, Graham Roberts

Department of Computer Science



Association (5)

- Aggregation/Compositions are treated in the same way but we may need to be careful about sharing objects.



18 Copyright © 2004, Graham Roberts

Department of Computer Science



Association (6)

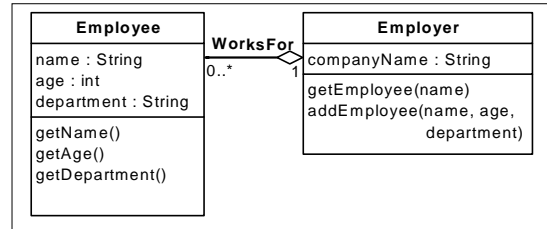
```
class MessageList
{
    private List messages =
        new ArrayList();
    ...
}
```

19 Copyright © 2004, Graham Roberts

Department of Computer Science



Associations (7)



20 Copyright © 2004, Graham Roberts

Department of Computer Science



Questions

21 Copyright © 2004, Graham Roberts

Department of Computer Science



Initialisation

- We have seen that constructors can be used to initialise instance variables.
- Both class and instance variables can also be directly initialised by initialisation expressions.

```
private int x = 2;
```

22 Copyright © 2004, Graham Roberts

Department of Computer Science



Initialisation (2)

- And also by an initialiser block:


```
private Stack x;
{ x = new Stack(); x.push(1); x.push(2);}
```
- A static initialiser block can be used for static variables.
- ```
private static Stack x;
static { x = new Stack(); x.push(1);
 x.push(2);}
```

23 Copyright © 2004, Graham Roberts

Department of Computer Science



## Choosing

- 3 ways to initialise - how do you choose?
- No single answer but:
  - aim to initialise a variable as close to the point of declaration as possible.
  - or group all initialisation into the constructor, so it is all in the same place.

24 Copyright © 2004, Graham Roberts

Department of Computer Science



## More than one constructor

- A class can have more than one constructor.
- Each can be used to initialise objects in a specific way.
- But won't all the constructors have the same name?
- Yes.

25

Copyright © 2004, Graham Roberts

Department of Computer Science



## Overloading

- Two or more methods or constructors can have the same name.
- But must have different arguments.
  - String()
  - String(byte[])
  - String(char[])
  - String(String)
  - String(byte[], int)

26

Copyright © 2004, Graham Roberts

Department of Computer Science



## Overloading (2)

- Return types are not considered:
  - int f(int)
  - float f(int) // Error
  - int f(int,int) // OK
  - float f(int, float) // OK
  - int f() // OK
- The compiler determines which method to call by matching the argument types.

27

Copyright © 2004, Graham Roberts

Department of Computer Science



## this

- **this** is special variable that is automatically declared in an instance method.
- It is a reference to the object the method was called for.
- Allows you to refer directly to the current object.

28

Copyright © 2004, Graham Roberts

Department of Computer Science



## this (2)

```
class T
{
 private Thing t;
 public int f(int x)
 {
 t.doSomething(this);
 }
}
```

29

Copyright © 2004, Graham Roberts

Department of Computer Science



## this (3)

```
class T
{
 private int x;
 public int f(int x)
 {
 this.x = x;
 }
}
```

30

Copyright © 2004, Graham Roberts

Department of Computer Science



## this (4)

- Can also be used to call a different overloaded constructor:

```
// This constructor does the real work
T(int x, int y, String z) { ... }
```

```
T() // Supply default values
```

```
{
 this(0,0,"Hello"); // no duplication of
} // init code
```



## Summary

- Looked at various details of the construction and use of classes.
- Overloading is a new variety of abstraction.
- Lots of details for the programmer to know about and use carefully.

