

G51PRG: Introduction to Programming Second semester Lecture 10

Natasha Alechina
School of Computer Science & IT
nza@cs.nott.ac.uk

Previous lecture: I/O

- I/O in Java
- Streams
- Reading, writing, handling exceptions
- Files
- Parsing

Lecture 10: revision

2

This lecture: revision using exercise 2

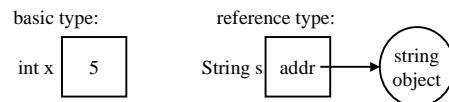
- Reference types (passing by reference vs by value)
- Extending classes, superconstructing, overriding methods
- Abstract classes
- Polymorphism
- Implementing interfaces

Lecture 10: revision

3

Reference types

- Objects are reference types
- Object type variables contain an address in memory of an object (not the object itself).
- This is different from basic types such as ints where variables actually contain a value.



Lecture 10: revision

4

Reference types: consequences

- `object1 == object2` means 'do `object1` and `object2` have the same address in memory?'
- `object1.equals(object2)` means are they equal, for example do they have the same values of all fields?
- For two Strings `string1` and `string2`, `string1.equals(string2)` means, do they have the same characters in the same sequence?
- `string1 == string2` means 'are `string1` and `string2` stored at the same address in memory?'

Lecture 10: revision

5

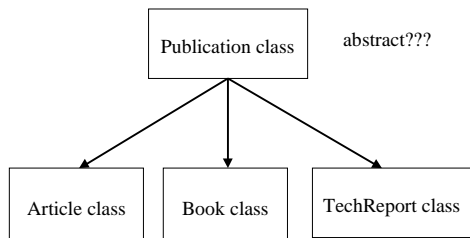
Exercise 2: checking for user input

```
String str = " ";
while (true) { // prompt and read
    UserInput.prompt("Enter Book, Article,
        TechReport or Exit: ");
    str = UserInput.readString();
    // test if the user wants to quit
    if (str.equals("Exit")) break;
    if (str.equals("Book")) {
        UserInput.prompt("Enter the author: ");
```

Lecture 10: revision

6

Exercise 2: class hierarchy



Lecture 10: revision

7

Publication class design

- Obvious things first: Publication class should have fields `String author`, `String title`, `String year`, and a constructor which sets them.
- Do we declare them as private? Depends on whether you want to access them directly (e.g. not via methods inherited from the parent or constructor of the parent) in the children.
 - If yes (you need to access `author` field in subclasses for example) - declare them protected.
 - If not, you may just as well declare them private.

Lecture 10: revision

8

Publication class: fields and constructor

```

// instance fields:
protected String author, title, year;
// constructor:
public Publication(String a, String t,
    String y) {
    this.author = new String(a);
    this.title = new String(t);
    this.year = new String(y);
}
// why not this.author = a; ?
  
```

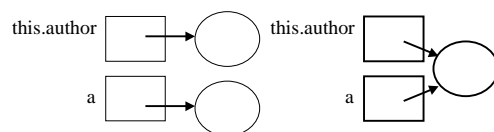
Lecture 10: revision

9

Reference types again

```

this.author = new String(a);
this.author = a;
  
```



Lecture 10: revision

10

Reference types: assignment

```

object1 = object2;
  
```

means: make **object1** refer to the same address in memory as **object2**.

If **object2** gets changed, so will **object1**, and vice versa (since they refer to the same object).

This may lead to unpredictable behaviour of the program.

Instead of assignment, it is safer to do cloning or duplication which involves allocation of new memory, as in

```

this.author = new String(a);
  
```

Lecture 10: revision

11

Publication class design continued

- Since Books, Articles and TechReports should all have `toHTML()` method, the best place to put it in is the parent Publication class.
- This does not guarantee that we can write the method once or even that we can write some useful part of the method, but we can use inheritance polymorphism: when we have an array or vector of mixed Publications, we can call `toHTML()` method on each of them without checking if it's a Book, an Article or a TechReport.

Lecture 10: revision

12

toHTML() in Publication class

- Options:
 - don't implement it; declare it abstract; declare Publication class abstract.
 - Implement a method which can be useful for at least some subclasses, for example Article and TechReport. They both require author and title fields in default font so parent class implementation can be used to provide the beginning of the string. The class can still be declared abstract if you like.
- Both options valid.

Lecture 10: revision

13

toHTML() in Publication class

- Since I am lazy I went for option 2:

```
public String toHTML() {
    return (author + title + ".");
}
```
- then in the TechReport and Article class we can do

```
public String toHTML() {
    return (super.toHTML() + " " +
            institution + " Technical Report " +
            number + ", " + year + ".");
}
```

, similarly for **Article**.

Lecture 10: revision

14

compareTo() in Publication class

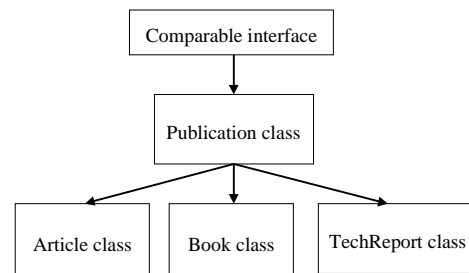
- Finally, we are supposed to have all publications implement Comparable. The natural thing is to get Publication to implement Comparable.
- So Publication should have

```
public int compareTo(Object p)
    method.
```

Lecture 10: revision

15

Exercise 2: class hierarchy



Lecture 10: revision

16

Implementing Comparable

```
public class Publication implements
    Comparable {
// fields
// constructor
// toHTML()
    public int compareTo(Object p) {
        return this.author.compareTo(
            ((Publication)p).author);
    }
} // use compareTo() method of Strings
```

Lecture 10: revision

17

Implementing interfaces in general

- Perhaps you noticed that Strings have both `String compareTo(String s)` and `Object compareTo(String s)` methods, which as you would expect do the same thing. String has been retrofitted to implement Comparable, just as Vector has been retrofitted to implement List.
- So the class itself did not get any extra functionality, but now Strings can be used in methods written for Comparable type objects.

Lecture 10: revision

18

Marker interfaces

- Some interfaces such as Cloneable are so called **marker interfaces** : they do not declare any methods (clone() is inherited from Object).
- A class implements the Cloneable interface to indicate to the Object.clone() method that it is legal for that method to make a field-for-field copy of instances of that class. Otherwise clone() throws CloneNotSupportedException.
- clone() can be overridden to produce a deep copy rather than shallow copy which Object.clone() does (it instantiates all fields of the new Object to the same things which the cloned object has, so they may share reference fields).

Lecture 10: revision

19

Subclasses

- Subclasses have extra fields;
- Need to write constructors; use superconstructing as in the previous exercise;
- Overwrite toHTML() (in Book completely, in Article and TechReport can use a bit of super.toHTML());
- Don't have to implement compareTo() in subclasses at all!

Lecture 10: revision

20

Example: Book class

```
public class Book extends Publication {
    String publisher;
    public Book(String a, String t, String
p, String y){
        super(a,t,y);
        this.publisher = new String(p);
    }
    public String toHTML() {
        return (author + ". " + "<i>" + title +
"</i>" + ". " + publisher + ", " + year
+ ".");
    }
}
```

Lecture 10: revision

21

Finally: Tester class main()

- declare a Vector or other List and some String variables
- loop while(true)
- ask for user input (Article, Book, TechReport, Exit)
- match it **using equals()**, **not** == to one of the four options
- if does not match anything, complain
- if matches Exit, break out of the loop
- if matches Book, prompt for 4 strings, pass them to Book constructor, create a Book object, insert it in the list
- similarly for Article and TechReport
- after the loop, call sort on the list
- print the list to the screen

Lecture 10: revision

22

Scheme of things

```
Vector list = new Vector();
String str = " ";
String a, t, y, p, j, v, n, i;
while (true) {
    // prompt for Article, Book etc.
    // read input into str
    // if does not match, complain
    if (str.equals("Exit")) break;
    if (str.equals("Book")) ...
}
// sort; print
```

Lecture 10: revision

23

If the user types "Book"

```
if (str.equals("Book")) {
    UserInput.prompt("Enter the author: ");
    a = UserInput.readString();
    UserInput.prompt("Enter the title: ");
    t = UserInput.readString();
    UserInput.prompt("Enter the publisher:
");
    p = UserInput.readString();
    UserInput.prompt("Enter the year: ");
    y = UserInput.readString();
    list.add(new Book(a,t,p,y));
}
```

Lecture 10: revision

24

Calling sort() and printing

```
Collections.sort(list);
for (int k = 0; k < list.size(); k++){
    System.out.println(
        ((Publication)list.elementAt(k)).toHTML()
    );
}
```

We can do this because of polymorphism; someone wrote a sort method which works for Publications just because they implement Comparable; we can keep various publications together because they are all Publications.

Lecture 10: revision

25

New exercise: Bibtex

- BiBTeX is a popular bibliography file format.
- The exercise is to read a bibtex file, parse it into entries, and produce a corresponding file in HTML with entries sorted alphabetically by author. If the file is in a wrong format, throw BadBibtexException.
- You may use the previous exercise, but do not have to.
- Extensions: style sheets (how html should look like); .bst files which bibtex uses may be a bit too complicated so feel free to define your own style formats. Relaxing the expected format of bibtex file, e.g. attributes in any order, upper or lower case...

Lecture 10: revision

26

Suggested method (don't have to!)

- Take a Bibtex file as an input;
- read the content of the file (to a String; see last lecture...)
- parse it into entries (see last lecture)
- create Publication objects from those entries, put them in some data structure, sort them (use the previous exercise)
- produce a String from the alphabetical list of publications in html and print it to output.html

Lecture 10: revision

27

How to submit

- Put all files in ~/Private/bibtex directory before the deadline.
- If not receive a confirmation email or receive an email that there was a problem, contact me.
- Generally don't give extensions; if you missed an exercise due to illness or other reasonable cause, get your tutor to write to me and I will set you alternative coursework (open in the last week of term, before the holiday, and open till the start of revision week).

Lecture 10: revision

28