MSc/ICY Software Workshop
Classes and Inheritance

Manfred Kerber www.cs.bham.ac.uk/~mmk

Tuesday 5 November 2019

Overview

1. Pocket calculator computations, base types, simple strings, variables, static methods, JavaDoc
   Wed/Thu/Fri: 1st Lab Lecture (login, editor, javac, javadoc)
2. Classes, objects, methods, JUnit tests
   Wed/Thu/Fri: 2nd Lab Lecture (Eclipse)
3. Conditionals, 'for' Loops, arrays, ArrayList
4. Exceptions, I/O (Input/Output)
5. Functions, interfaces
6. **Sub-classes, inheritance, abstract classes**
7. Inheritance (Cont'd), packages
8. Revision
9. Graphics
10. Graphical User Interfaces
11. Graphical User Interfaces (Cont'd)

Changes possible

Object-Oriented Programming

Distinguish:

- **Classes, e.g.,** `BankAccount`, `Customer`
- **Objects, e.g.,** `bankAccountJohn`, `customerMary`
  created by a Constructor, e.g.
  `public BankAccount (Customer customer, String password)`
- **Methods, e.g.** `getBalance()`

Superclass vs subclass

- A subclass `SubclassA` inherits from its (unique) superclass `SuperclassB` (introduced by `public class SubclassA extends SuperclassB`)
- All methods not explicitly overridden in the subclass are inherited from the superclass.
- Overridden methods from the superclass are accessible via `super` in the body of the overriding method, e.g., in writing the code for a `toString()` method you can use `super.toString()`.
- Variables (and methods) private to the superclass are not accessible from the subclass.

Rationale for Inheritance

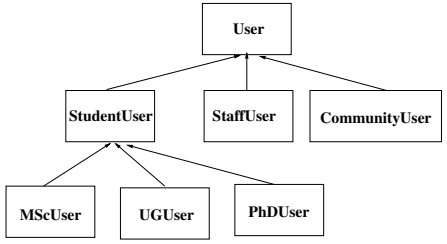
Inheritance is a very important feature of object oriented programming.

- Inherit methods, that is, methods that are common to the superclass can be used without duplication of code.
- Inheritance keeps code simpler.
- If code needs to be changed (remember, typically changes are frequent) then it can be changed at a single point. This is important since it makes code maintainable (if code is duplicated any changes may be messy since parts may be overlooked and inconsistencies may be introduced).

Example: rudimentary library system

- The library offers books on loan, either on `shortLoan` (one day) or `longLoan` (at most 30 days)
- `User`s are either `StudentUser`, `StaffUser`, or `CommunityUser`.
- Assume that students can borrow at most 10 books, staff and members of the community as many as they like. Members of the community have to pay a fee of £ 1 per book (others not). `UGUser`s can borrow books for at most 10 days, `MScUser`s for at most 20 days, `PhDUser`s for at most 30 days.

Hierarchy of super- and subclasses



Abstract Class

E.g., `public abstract class User`.
Abstract classes do not have immediate objects, but only via subclasses.
With an abstract class `User`, with an abstract subclass `StudentUser` and (non-abstract) subclasses `StaffUser`, and `CommunityUser`
as well as the three (non-abstract) subclasses of `StudentUser`:

- `UGUser`,
- `MScUser`, and
- `PhDUser`

each user object generated is member of one of the classes `UGStudent`, `MScStudent`, `PhDStudent`, `StaffUser`, or `CommunityUser`.

Classes

- have an abstract class `User`
- distinguish three subclasses of users: `StudentUser`, `StaffUser`, `CommunityUser`, using inheritance.
- distinguish three subclasses of `StudentUser`: `UGUser`, `MScUser`, `PhDUser`, using inheritance.
- For a `User` we know their `firstName`, `surname`, `phoneNumber`, `booksOnLoan`. Each `bookOnLoan` goes with the `Book`, the `DateTime` when it was borrowed, and the `DateTime` when it has to be given back.

Build suitable classes: `User`, `StudentUser`, `StaffUser`, `CommunityUser` making use of inheritance to model the situation.

Class Membership

In order to check whether an object belongs to a particular class you can use
`object instanceof class`. In such a case you can also cast the type of the object to this type by `(type) object`.
For instance, with `User u` you can check:
`u instanceof StudentUser`. If this is the case you can cast the type of `u` to `StudentUser` by `(StudentUser) u`