CSSE2002/7023

Semester 1, 2021

Programming in the Large

Week 3.1: More Inheritance and Things Going Wrong

In this Session

- Interfaces
- `super`
- Duck Typing (Python vs Java)
- Casting
- Consequences of Java's Design
- Handling Errors

Quick Recap – Classes and Objects

- Class
- Object

Quick Recap – Classes and Objects

- Class – describes the behaviour and state of a type.
- Object

Quick Recap – Classes and Objects

- Class – describes the behaviour and state of a type.
- Object – is an instance of a class. It stores one instance of the state that the class describes. Behaviours described by the class are applicable to the object / instance.

Quick Recap – Inheritance Access

```
public class X {
  public int one;
  private int two;
  protected String three;

  public X() {...}
  private X(int arg) {...}

  public int f() {...}
  private int g() {...}
  protected int h() {...}
}
```

```
public class Y
    extends X {

}
```

```
public class Z {

}
```

More Inheritance

`class Child extends Parent` ⇒

`Child` gets the following from `Parent`

1. (public/protected) methods and variables
   - what it says it can do
2. Method bodies/implementation
   - how it does it

More Inheritance

`class Child extends Parent` ⇒

`Child` gets the following from `Parent`

1. (public/protected) methods and variables
   - what it says it can do
2. Method bodies/implementation
   - how it does it

Sometimes all we care about is what public methods are present in a class (not trying to inherit code).

- Only #1

Interfaces

An `interface` is a type like a class but it contains no method bodies.

e.g. `java.lang.Comparable`
declares a single method, `compareTo` which takes an object and returns an `int` $\in \{< 0, 0, > 0\}$.

So any code which takes objects and will need to order them, could specify them as being of type `Comparable`.

e.g. `int doStuff(Comparable c[])`

Interfaces

To declare that your class will implement an interface use `implements`.

```
public class Duck extends Fowl
        implements Comparable, Clonable
```

`Duck` inherits code and members from the `Fowl` class.

and

Has all the methods which `Comparable` and `Clonable` say should be there.

Interfaces

`InterfaceExample.java`

- A class implementing an interface is responsible for supplying method bodies for everything declared in the interface.
- Could be used to advertise that your class has additional useful capabilities.
- Could be used to indicate that your class belongs to multiple groups. (e.g. Someone is both a staff member and a student.)
- `interfaces` fill a role taken by abstract[1] classes in other languages.
    - Necessary because Java only allows extending from a single class[2].

[1] which Java also has

[2] "single inheritance" – Python allows multiple inheritance

super

SquareN.java

- Use of super, shadowing and this

No Ducks

Python will let you write code to access a member without knowing whether it actually exists — it checks at runtime.

Java won't let you try to access something unless it is sure (at compile time) that it exists.

```
Object obj = "Hello";
int l1 = obj.length(); // compile error
                       // Object has no .length()

Object obj = "Hello";
String s1 = (String)obj;
int l1 = s1.length();
```

OR

```
Object obj = "Hello";
int l1 = ((String) obj).length();
```

Casting

`(newtype) oldvalue` – called a “typecast” or just “cast”.

- Tells the compiler that you want a *newtype* value of *oldvalue*.

```
(int) 1.7
long x = 100; (float) x
```

Some casts will be made automatically (implicitly)

- “smaller” types $\rightarrow$ “larger” types.
    - e.g. `int` to `long`, `float` to `double`
- integer types $\rightarrow$ floating point types.
    - e.g. `int` to `float`

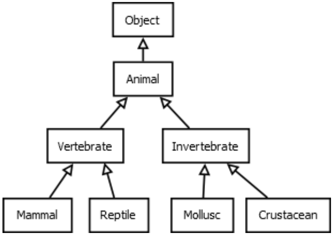
Casting a variable does not change the value of the variable.

Casting References

`(String) obj`

- Returns a reference which can be used to access `String` members in `obj`.
- Changes the compiler's view of what it is looking at.
- It does not change `obj` itself.
- If the compiler does not know of an inheritance relationship between the known type of `obj` and the cast type, it will not compile.
- At runtime, Java will check to see if the cast is valid.
    - If it isn't, it will throw a `ClassCastException`.

Casting References



From	**To**	**Implicit?**	**Error?**
Vertebrate	Animal	yes	—
Vertebrate	Mammal	no	runtime?
Vertebrate	Object	yes	—
Vertebrate	Mollusc	no	compile

“Upcasts” can be implict, “Downcasts” can’t.

You cannot cast between primitives and objects.
e.g. `(String) 5;` is no bueno.

Consequences of Java's Design

- All callable code is in methods
  - No free functions
  - Greater use of "`static`" than in other languages
  - More implementation/non-model classes (e.g. for call backs)
- All variables and constants must be in classes or methods.
- Primitive types are not objects:
  - Containers only hold objects
  - Need wrapper classes for some things
- Single inheritance means interfaces are needed to declare extra capabilities into hierachy

When It All Goes Wrong

Breaking code into functions means we need to work out what to do if something fails.

- Things can fail? Surely I'd notice.

When It All Goes Wrong

Breaking code into functions means we need to work out what to do if something fails.

- Things can fail? Surely I'd notice.
- Silent failure — detect, and avoid doing something silly

When It All Goes Wrong

Breaking code into functions means we need to work out what to do if something fails.

- Things can fail? Surely I'd notice.
- Silent failure — detect, and avoid doing something silly
- Print to console

When It All Goes Wrong

Breaking code into functions means we need to work out what to do if something fails.

- Things can fail? Surely I'd notice.
- Silent failure — detect, and avoid doing something silly
- Print to console
    - Fine for quick debugging

When It All Goes Wrong

Breaking code into functions means we need to work out what to do if something fails.

- Things can fail? Surely I'd notice.
- Silent failure — detect, and avoid doing something silly
- Print to console
  - Fine for quick debugging
  - Hard to process from code

When It All Goes Wrong

Breaking code into functions means we need to work out what to do if something fails.

- Things can fail? Surely I'd notice.
- Silent failure — detect, and avoid doing something silly
- Print to console
    - Fine for quick debugging
    - Hard to process from code
- `boolean` return

When It All Goes Wrong

Breaking code into functions means we need to work out what to do if something fails.

- Things can fail? Surely I'd notice.
- Silent failure — detect, and avoid doing something silly
- Print to console
  - Fine for quick debugging
  - Hard to process from code
- `boolean` return
- error code return

When It All Goes Wrong

Breaking code into functions means we need to work out what to do if something fails.

- Things can fail? Surely I'd notice.
- Silent failure — detect, and avoid doing something silly
- Print to console
  - Fine for quick debugging
  - Hard to process from code
- `boolean` return
- error code return
  - `int` indicating what went wrong. [Use static constants for this]

When It All Goes Wrong

Breaking code into functions means we need to work out what to do if something fails.

- Things can fail? Surely I'd notice.
- Silent failure — detect, and avoid doing something silly
- Print to console
  - Fine for quick debugging
  - Hard to process from code
- `boolean` return
- error code return
  - `int` indicating what went wrong. [Use static constants for this]
  - Make sure you include one for success

When It All Goes Wrong

Breaking code into functions means we need to work out what to do if something fails.

- Things can fail? Surely I'd notice.
- Silent failure — detect, and avoid doing something silly
- Print to console
    - Fine for quick debugging
    - Hard to process from code
- `boolean` return
- error code return
    - `int` indicating what went wrong. [Use static constants for this]
    - Make sure you include one for success
    - Variants: String, enums

Need to Get an Answer Out as Well?

Need to Get an Answer Out as Well?

- Sentinel — Special value which is not meant to be taken literally.

Need to Get an Answer Out as Well?

- Sentinel — Special value which is not meant to be taken literally.
    - e.g. indexOf() $== -1$ or position
        - Object or `null`

Need to Get an Answer Out as Well?

- Sentinel — Special value which is not meant to be taken literally.
    - e.g. indexOf() $== -1$ or position
        - Object or `null`
    - Be careful that the sentinel is never a valid value. (See Y2K)

Need to Get an Answer Out as Well?

- Sentinel — Special value which is not meant to be taken literally.
  - e.g. indexOf() $== -1$ or position
    - Object or `null`
  - Be careful that the sentinel is never a valid value. (See Y2K)
  - May need a separate method to get details of the failure

Need to Get an Answer Out as Well?

- Sentinel — Special value which is not meant to be taken literally.
    - e.g. indexOf() $== -1$ or position
        - Object or `null`
    - Be careful that the sentinel is never a valid value. (See Y2K)
    - May need a separate method to get details of the failure
- Return a status object containing results or failure information

Need to Get an Answer Out as Well?

- Sentinel — Special value which is not meant to be taken literally.
    - e.g. indexOf() $== -1$ or position
        - Object or `null`
    - Be careful that the sentinel is never a valid value. (See Y2K)
    - May need a separate method to get details of the failure
- Return a status object containing results or failure information
- Extra information out via OUT or reference params
    - *Not applicable to Java*

Need to Get an Answer Out as Well?

- Sentinel — Special value which is not meant to be taken literally.
  - e.g. indexOf() $== -1$ or position
    - Object or `null`
  - Be careful that the sentinel is never a valid value. (See Y2K)
  - May need a separate method to get details of the failure
- Return a status object containing results or failure information
- Extra information out via OUT or reference params
  - *Not applicable to Java*

Everything so far has used the return value to get info out.