More Java OO

CSC207 Winter 2015

Inheritance hierarchy

All classes form a tree called the inheritance hierarchy, with `Object` at the root.

Class `Object` does not have a parent. All other Java classes have one parent.

If a class has no parent declared, it is a child of class `Object`.

A parent class can have multiple child classes.

Class `Object` guarantees that every class inherits methods `toString`, `equals`, and others.

Inheritance

Inheritance allows one class to inherit the non-private data/ methods of another class.

In a subclass, `super` refers to the part of the object defined by the parent class.

To call a parent's constructor:

```
super(arguments);
```

A subclass can add new data and methods.

A subclass can also reuse a name already is used for an inherited data member or method. More to come on the implications of this...

Shadowing and Overriding

Suppose class `A` and its subclass `AChild` each have

- an instance variable `x`
- an instance method `m`

`A`'s `x` is **shadowed** by `AChild`'s `x`.

- This is confusing and rarely a good idea.

`A`'s `m` is **overridden** by `AChild`'s `m`.

- This is often a good idea. We often want to specialize behaviour in a subclass.

If a method must not be overridden in a descendant, declare it `final`.

Dynamic Binding

A variable can be assigned objects of its type or any subtype.

As we saw, which methods are called depend not on the variable type, but on the type of the object it currently refers to.

Overriding and dynamic binding allow a superclass to define some things that it and all of its descendants share, and yet the correct specialized method to be called.

Casting Down

```
Object o = new String("hello");
```

We cannot do the following, because `Object` does not have a `charAt` method:

```
char c = o.charAt(1);
```

Instead, we need to cast the object that `o` refers to as a `String`:

```
char c = ((String) o).charAt(1);
```