Lets Code!

Inheritance

Inheritance

What we'll cover

What we'll cover

Class inheritance

What we'll cover

Class inheritance

Object Class

What we'll cover

Class inheritance

Object Class

Object Methods

What we'll cover

Class inheritance

Object Class

Object Methods

Super and Final Keywords

Terminology

- Subclass - a class that inherits some of its behavior from another class
- Superclass - the class from which a subclass inherits
- Parent/child class - synonyms for superclass and subclass; slightly imprecise

Class inheritance

- classes can inherit the interfaces of classes and extend their feature set
- this is achieved with the `extends` keyword
- If `B` extends `A` then `B` has all of `A`'s public and protected members and methods
  - package access only within the same package

Inheritance is an "is a" or "is like a" relationship

- An SUV **is a** Vehicle
- A corgi **is a** Dog

Example: SUV

```
public class Vehicle{
  public void start(){...}
}

class SUV extends Vehicle{
  public void drive(){
    start();
    ...
  }
}
```

Example: Corgi

```
class Dog{public void wag(){...}}
public class Corgi extends Dog{
  public static void main(String[] args){
    Corgi bucket = new Corgi();
    bucket.wag();
  }
}
```

Full example

Upcasting

- Objects can be treated as instances of any superclass in the class hierarchy

```
public class App{
 public static void main(String[] args){
  Dog pembroke = new Corgi();
  pembroke.wag();
  }
}
```

All classes inherit from the Object class

- `Object` is the immediate superclass if `extends` is omitted

Extending Object class

- valid but redundant
- ...unless you've defined an `Object` class of your own (please don't)

Example: Explicit "extends Object"

Both the same:

```
class Thing {}
```

```
class Thing extends Object {}
```

Object Methods

- `equals()`, `hashCode()`, and `toString()`

equals

- `equals(Object o)` checks equality between this object and Object `o`
- Default implementation returns true only if this object *IS* Object `o`
- Mutable objects should keep default implementation, immutable objects may override
- Overriding `equals` is dangerous and tricky

hashCode

- returns a nearly-unique hash (number) representing the object
- used in `HashSet`, `HashMap` and other hashing data structures
- closely related to `equals()` - equal objects must have the same hash code

toString

- Produces a string representation of the object
- default is `m.getClass().getName() + "@" + Integer.toHexString(m.hashCode())`
- automatically called by `System.out` methods
- Override `toString` for a readable way to print your objects.

Super keyword

- Behaves like `this` but treats the current object like its superclass
- Unlike `this`, `super` is not a reference

Abstract classes

- Cannot be instantiated
- Can contain abstract methods
- Can extend other abstract classes

Abstract methods

A method whose signature is defined, but implementation is not

```
abstract class Worker{

  // All workers can doWork, but some do it differently than others
  public WorkProduct doWork();
}
```

Extending Abstract Classes

- Inheritance works the same as concrete classes
- All abstract methods are inherited
- Abstract methods can be implemented
- Concrete classes must implement all remaining abstract methods

Interfaces vs abstract classes

- Interfaces can be multiply implemented
- Abstract classes have state and private fields and methods available
- Concrete classes extend abstract classes; they implement interfaces (syntax difference)
- Abstract classes can implement interfaces; the reverse is not allowed

The keyword: `final`

- prevents values and references from changing after initialization
- prevents inheritance of classes
- prevents overriding of methods (but not overloading)
- `static final` produces a compile-time constant

Blank final fields

- A field can be marked final but not initialized until the constructor runs
- All constructors must initialize blank finals, or it is an error.

```
public class Foo{
  final int x;
  public Foo(){
    x = 4;
  }
}
```

final pitfalls

- final object references cannot be changed but the underlying object can
- blank final fields must be initialized in every constructor
- `final` != `finally` -- these are two different keywords