Inheritance and Interfaces

- Inheritance models an "is-a" relationship
  - A dog is a mammal, an TreeSet is a Set which is a Collection, a square is a shape, …
- Write general programs to understand the abstraction, advantages?

```
void doShape(Shape s) {
  System.out.println(s.area());
  System.out.println(s.perimeter());
  s.expand(2.0);
}
```

- But a dog is also a quadruped, how can we deal with this?

Single inheritance in Java

- A class can extend only one class in Java
  - All classes extend Object --- it's the root of the inheritance hierarchy tree
  - Can extend something else (which extends Object), why?
- Why do we use inheritance in designing programs/systems?
  - Facilitate code-reuse (what does that mean?)
  - Ability to specialize and change behavior
    - If I could change how method `foo()` works, `bar()` is ok
  - Design methods to call ours, even before we implement
    - Hollywood principle: don't call us, …

Guidelines for using inheritance

- Create a base/super/parent class that specifies the *behavior* that will be implemented in subclasses
  - Subclasses specify inheritance using `extends Base`
- Inheritance models "is-a" relationship, a subclass is-a parent-class, can be used-as-a, is substitutable-for
  - Standard examples include animals and shapes
- OOP Terminology
  - *Hierarchy*: classes are arranged like a tree, with superclasses appearing above its subclasses
  - *Overriding*: When an object receives a message, it checks its own methods first before consulting its superclass.
  - *Polymorphism*: method binding is determined at *run-time*

Student behavior

```
public class Student
{
  private String myName;
  protected int myEnergy;

  public Student(String name)

  public String getName()
  public int getEnergy()

  public boolean isAlive()
  public void eat()
  public void work()
  public void live()
  // …
```

Implementation of behavior

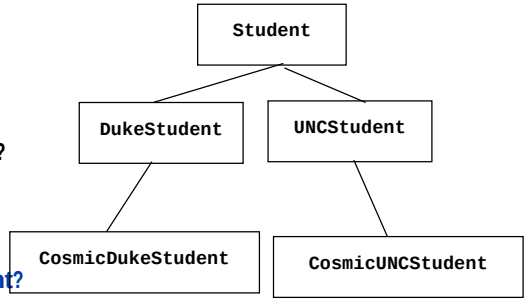
```
public void sleep()
{
  myEnergy += 10;
  System.out.println("Zzzzzzzzzzzz, resting sleep”);
}

public void live()
{
    eat();
    work();
    sleep();
}
```


See Student.java, School.java

- How do subclass objects call parent class code, see `DukeStudent` class in School.java
  - `super` syntax
- Why is base class data protected rather than private?
  - Must be accessed directly in subclasses, why?
  - Not ideal, try to avoid state in base/parent class: trouble
    - What if derived class doesn't need data?

Difference in behavior?

- What's a field and what's a method?
  - # tires on car?
  - # doors on car?
  - How student lives?
- Where does name of school belong? What about energy increment?
- What's problem with hierarchy here?
  - NCState student?



Problems with inheritance

- Consider the student example and burrito eating
  - CosmicStudent is a subclass of DukeStudent
    - What behavior changes in the new subclass?
  - What about a UNCStudent eating cosmic cantina food?
    - Can we have CosmicDukeStudent and CosmicUNCStudent?
    - Problems with this approach?
- Alternative to inheritance: use delegation (aka layering, composition)
  - Just like myEnergy is a state variable with different values, make myEater a state variable with different values
  - Delegate behavior to another object rather than implementing it directly

Delegation with school/student

- If there's a class Eater, then what instance variable/field will a Student store to which eating behavior delegated?

```
public void eat()
{
    myEater.doEat();
}
```

  - How is the eater instance variable initialized?
  - Could we adopt this approach for studying too?
  - When is this approach better/worse?

Multiple Interfaces

- Classes (and interfaces) can implement multiple interfaces
  - A dog is a mammal, a quadruped, a pet
  - How come canine is different?
  - What behavior do quadrupeds have? Pets have?

- An interface specifies the name (and signature) of methods
  - No implementation, no state/fields
  - Yes for constants

Comparable and Comparator

- Both are interfaces, there is no default implementation
  - Contrast with `.equals()`, default implementation?
  - Contrast with `.toString()`, default?

- Where do we define a Comparator?
  - In its own .java file, nothing wrong with that
  - Private, used for implementation and not public behavior
    - Use a nested class, then decide on static or non-static
    - Non-static is part of an object, access inner fields

- How do we use the Comparator?
  - Sort, Sets, Maps (in the future)