1. Log in
2. Today We Begin Unit 5
   - Inheritance (overriding and more)
   - Polymorphism
   - Abstract Classes
   - Interfaces

Oct 16-7:52 AM

Examine our current Cat class ...

```
public class Cat extends Animal{
   public boolean meowing;
   public Cat(){
      super();
      meowing=false;
   }
   public void makeMeow(){
      meowing=true;
   }
   public void stopMeow(){
      meowing=false;
      super.goToSleep();
   }
   public boolean getMeowStatus(){
      return meowing;
   }
}
```

Nov 11-3:13 PM

Cat class can borrow death( )

Since it has inherited Animal ...

```
public void death( ) {
   living=false;
}
```

```
public class Cat extends Animal{
   public boolean meowing;
   public Cat(){
      super();
      meowing=false;
   }
   public void makeMeow(){
      meowing=true;
   }
   public void stopMeow(){
      meowing=false;
      super.goToSleep();
   }
   public boolean getMeowStatus(){
      return meowing;
   }
}
```

Nov 11-3:13 PM

But, doesn't a cat have 9 lives???

```
public void death( ) {
   living=false;
}
```

*** We should override death( ) ***

*** A cat should have a different method ***

```
public class Cat extends Animal{
   public boolean meowing;
   public Cat(){
      super();
      meowing=false;
   }
   public void makeMeow(){
      meowing=true;
   }
   public void stopMeow(){
      meowing=false;
      super.goToSleep();
   }
   public boolean getMeowStatus(){
      return meowing;
   }
}
```

Nov 11-3:13 PM

First, every cat needs 9 lives ...

Need a new instance variable for cat objects. (track number of lives)

```
public class Cat extends Animal{
   public boolean meowing;
   public int lives;
   public Cat(){
      super();
      meowing=false;
      lives=9;
   }

}
```

Nov 11-3:13 PM

First, every cat needs 9 lives ...

Need a new instance variable for cat objects. (track number of lives)

Now every cat that gets created starts with 9 lives!

```
public class Cat extends Animal{
   public boolean meowing;
   public int lives;
   public Cat(){
      super();
      meowing=false;
      lives=9;
   }

}
```

Nov 11-3:13 PM

We want a death( ) method to override the inherited death( )...

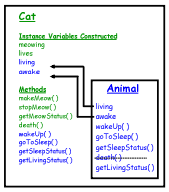
@Override

This method will now run for all Cat objects instead of the Animal death( ) method!

```
public class Cat extends Animal{
   public boolean meowing;
   public int lives;
   public Cat(){
      super();
      meowing=false;
      lives=9;
   }
   ...
   @Override              //capital O
   public void death( ){
      this.lives--;
      if(this.lives==0)
         this.living=false;
   }
}
```

Nov 11-3:13 PM

Cat class review ...




```
public class Cat extends Animal{
   public boolean meowing;
   public int lives;
   public Cat(){
      super();
      meowing=false;
      lives=9;
   }
   public void makeMeow(){
      meowing=true;
   }
   public void stopMeow(){
      meowing=false;
      super.goToSleep();
   }
   public boolean getMeowStatus(){
      return meowing;
   }
   @Override
   public void death(){
      this.lives--;
      if(this.lives==0)
         this.living=false;

   }
}
```

Nov 11-3:13 PM

Review of subclasses

1. They can add new instance variables
2. They can add new methods
3. They can use inherited methods
4. They can override inherited methods
5. They cannot change public things to private
6. They cannot change inherited static methods
7. They should have their own constructors
8. They cannot access inherited private data without the aid of a method

Nov 12-2:53 PM

Creating various objects with our classes ...

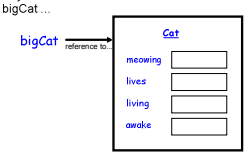
Cat bigCat= new Cat( );

Nov 12-3:33 PM

Creating various objects with our classes ...

Cat bigCat= new Cat( );

Create a cat Object called bigCat ...

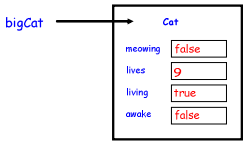


Nov 12-3:33 PM

Creating various objects with our classes ...

Cat bigCat= new Cat( );

Fill it using the constructor method Cat( )...



Nov 12-3:33 PM

Technically the following works, but ...

Animal smallCat= new Cat( );

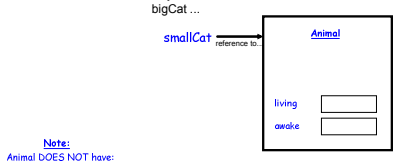
*** Create a new Animal called smallCat using the Cat Constructor ***

Nov 12-3:33 PM

Creating various objects with our classes ...

Animal smallCat = new Cat( );

Create an animal Object called bigCat ...

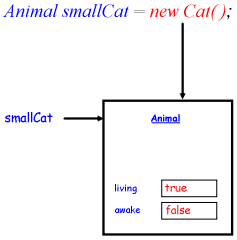


Note:
Animal DOES NOT have:
meowing
lives

Nov 12-3:33 PM

Creating various objects with our classes ...

Animal smallCat = new Cat( );

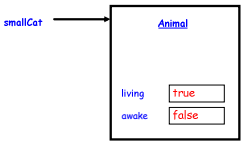


Note:
Animal DOES NOT have:
meowing
lives

Nov 12-3:33 PM

Notice, this could be a problem ...

Animal smallCat = new Cat( );



smallCat.living //acceptable code
smallCat.lives //NOT VALID, smallCat is an Animal

*** Cat( ) constructor may have been used but this is an Animal Object ***

Nov 12-3:33 PM

There is a way though ... lets review Casting ...

Remember: (int)(3.47) = 3
(int)(4.98) = 4
(double)(5) = 5.0
...

Casting: Forcing something to become something else!

Nov 12-3:33 PM

In other words ...

(int)(6.8773238)

make this
become

Nov 12-3:33 PM

In other words ...

(int)(6.8773238)

this ← make this become

Nov 12-3:33 PM

Casting: making something become something else ...

Nov 12-3:33 PM

Casting: making something become something else ...

I want my Animal to become a Cat so I can make it meow!

Nov 12-3:33 PM

Casting: making something become something else ...

I want my Animal to become a Cat so I can make it meow!

(Cat)(Animal)

this ← make this become

Nov 12-3:33 PM

Casting: making something become something else ...

I want my Animal to become a Cat so I can make it meow!

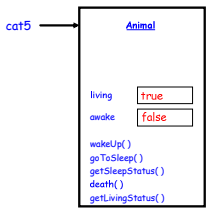
(Cat)(animalName)

this ← make this become

Nov 12-3:33 PM

Example:

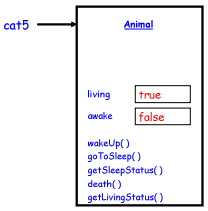
Animal cat5 = new Cat(); *//Build an Animal using Cat constructor*



Nov 12-3:33 PM

Example:

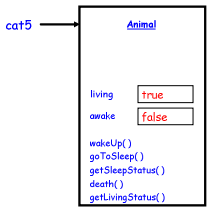
```
Animal cat5 = new Cat();          //Build an Animal using Cat constructor
System.out.println(cat5.living);  //works ... animal=living
cat5.getSleepStatus();            //works ... animal
```



Nov 12-3:33 PM

Example:

```
Animal cat5 = new Cat();          //Build an Animal using Cat constructor
System.out.println(cat5.living);  //works ... animal=living
cat5.getSleepStatus();            //works ... animal
cat5.lives;                       //DOES NOT WORK, this is an animal
```



Nov 12-3:33 PM

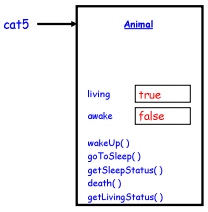
Example:

```
Animal cat5 = new Cat();          //Build an Animal using Cat constructor
System.out.println(cat5.living);  //works ... animal=living
cat5.getSleepStatus();            //works ... animal
cat5.lives;                       //DOES NOT WORK, this is an animal
```

... but we want to cast cat5 into a cat!

(Cat)(animalName)

this ← make this become



Nov 12-3:33 PM

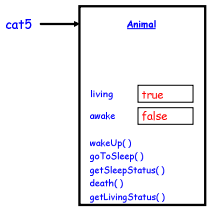
Example:

```
Animal cat5 = new Cat();          //Build an Animal using Cat constructor
System.out.println(cat5.living);  //works ... animal=living
cat5.getSleepStatus();            //works ... animal
cat5.lives;                       //DOES NOT WORK, this is an animal
((Cat)(cat5)).lives;              //WORKS! cat5 is now a cat!
```

... but we want to cast cat5 into a cat!

(Cat)(cat5)

this ← make this become

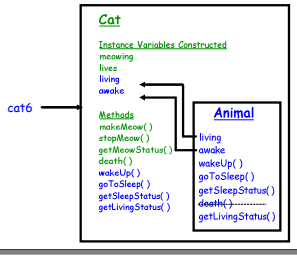


Nov 12-3:33 PM

Now Analyze this code:

```
Animal cat5 = new Cat();          //Build an Animal using Cat constructor
System.out.println(cat5.living);  //works ... animal=living
cat5.getSleepStatus();            //works ... animal
cat5.lives;                       //DOES NOT WORK, this is an animal
System.out.println(((Cat)cat5).getMeowStatus());
System.out.println(((Cat)cat5).lives);
Cat cat6 = ((Cat)cat5);
System.out.println(cat6.lives);
```

*** This is tricky so practice a little ***



Nov 12-3:33 PM

Things to do ...

1. Wrap Up Unit 5 WS 01-02
2. Work on Unit 5 WS03 More on Inheritance

Oct 16-9:12 AM