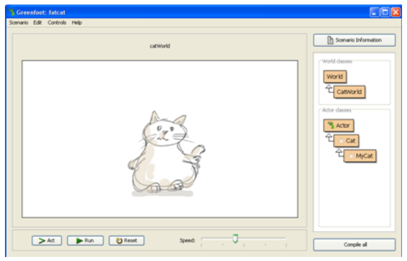
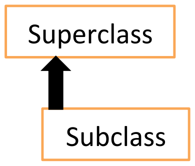
Fat Cat Simulation

Based on Games and Simulations
Thomas Cooper - The Walker School

Start Fat Cat Project

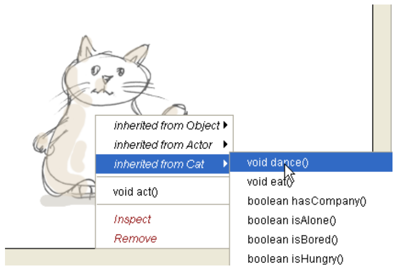
- Open the fatcat scenario
  - In extra scenarios folder
    - You may need to click “Compile all”
- The right side shows the UML class diagram
  - Which classes are subclasses?





Invoking Methods

- Right click on the cat and move the mouse to the right arrow.
  - Click on a method to execute it.
    - Try out the different methods
  - How many methods exist on myCat?
  - How many are inherited from Cat and from Actor?



Creating Classes in Java

- Each public class is created in its own file
  - The filename must match the class name and end in “.java”: Cat.java
  - This is what a person can read and change
- The code is <u>compiled</u> into a file that ends in ".class"
  - This is what a computer can read and execute
- Find the .java and .class files on your computer.
  - Look <u>outside</u> Greenfoot in ‘My Computer’ or Finder. Browse to the greenfoot scenarios - extra scenarios - fat-cat start folder. You should see many files.

The Java Class Structure – Classes look like this

import package.class; // as needed
public class *Name* extends *OtherName*
{
  // fields – the data for each object
  // constructors – initialize the fields
  // methods – behavior for the objects
  // optional main method for testing
}

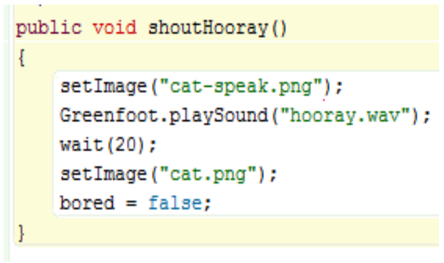
Blocks in Java Right-click on myCat and Open editor

- The '{' and '}' come in pairs.
  - Each pair defines the beginning and end of a block
- A class is defined in a block

  public class MyCat extends Cat
  { // starts the class
  } // ends the class
- There is a block of code that defines a method as well.
  - Notice the act() method also has a block in it.

Statements in Java – Open the editor for the Cat class, find shoutHooray()

- Statements do some action
  - Like setting an image
  - Or playing a sound
- Always end in ';'
- Are inside of method blocks
  - Indented to show that they are in the block, like Python!



```
public void shoutHooray()
{
    setImage("cat-speak.png");
    Greenfoot.playSound("hooray.wav");
    wait(20);
    setImage("cat.png");
    bored = false;
}
```

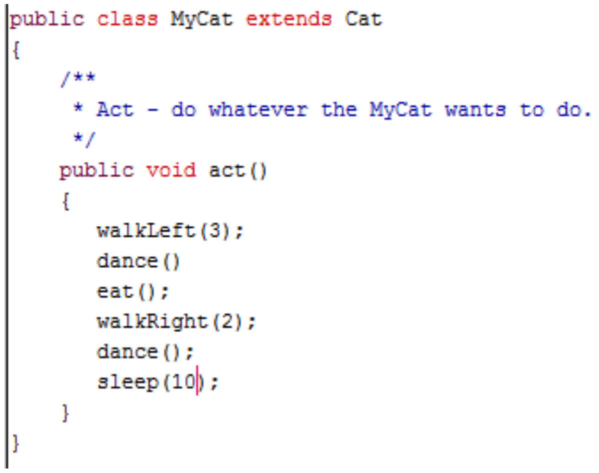
Challenge – Write some code!

- Modify the act method in MyCat by adding method calls
- Compile all and try it out

public void act()
{
  // add code here
}

- Have it dance and then walk left and then eat
- Create your own set of things for it to do when it acts
- Notice that a subclass can *override* a method of the superclass
  - act

One Possible Solution

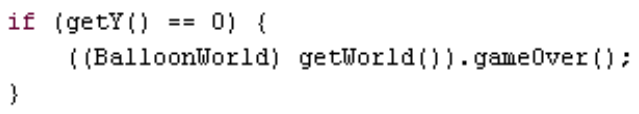


```
public class MyCat extends Cat
{
    /**
     * Act - do whatever the MyCat wants to do.
     */
    public void act()
    {
       walkLeft(3);
       dance()
       eat();
       walkRight(2);
       dance();
       sleep(10);
    }
}
```

Conditional Execution

- You can use the keyword ‘if’ to conditionally execute a statement or block of statements
  - When a Boolean expression is true
  - A Boolean expression is one that is true or false
- If statements look like this in Java –

```
if (something == TRUE) {
     statement;
}                              Notice the brackets!
```



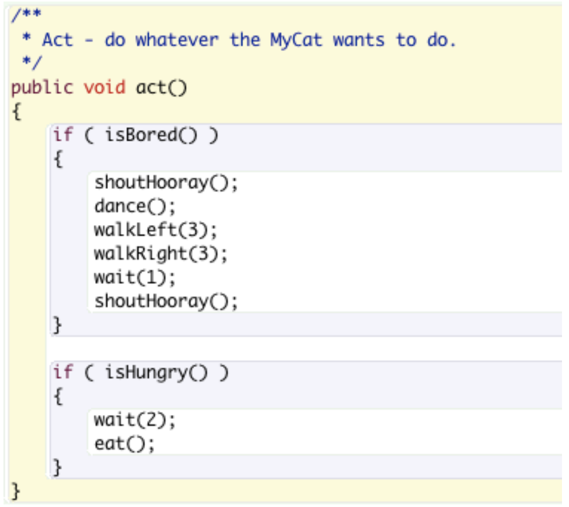
```
if (getY() == 0) {
    ((BalloonWorld) getWorld()).gameOver();
}
```

Challenge

- Modify the MyCat act method
  - To sleep when tired
  - To eat when hungry
  - To dance when bored
- Use a conditional statement for each of these
  - Hint: Use an ‘if’ like this -

```
if ( isSleepy() )
{
    do what???  Code here…
}
```

One Possible Solution – Add if(isSleepy())



```
/**
 * Act - do whatever the MyCat wants to do.
 */
public void act()
{
    if ( isBored() )
    {
        shoutHooray();
        dance();
        walkLeft(3);
        walkRight(3);
        wait(1);
        shoutHooray();
    }

    if ( isHungry() )
    {
        wait(2);
        eat();
    }
}
```