CSE 113

September 20, 2010

Announcements

- Lab 3 posted
- Grades for all labs are still not being computed because of problem with Web-CAT's grading module – hopefully we will get to a resolution this week
- Exam 1 Review – Wednesday (review sheet posted on website)
- Exam 1 Friday

Car Example from class

- Let's look at the code for the Car class and discuss parts of it.
- I've added stuff as well.

```
public class Car extends Vehicle {
   public void act() {
       if(atWorldEdge()) {
           turn(15);
       }
       if(Greenfoot.getRandomNumber(100) < 20) {
           turn(Greenfoot.getRandomNumber(181) – 90);
       }
       if(Greenfoot.isKeyDown("left")) {
           turn (-36);
       }
       move();
       if(canSee(Barrel.class)) {
           destroy(Barrel.class);
       }
   }
}
```

What happens every time act is called?

- First, we check if we are at the edge of the world. If so, we turn.
- Then, get a random number. If it is less than 20, we turn again.
- Third, check if the user has pushed the left arrow key, if so, we turn.
- Fourth, we move.
- Fifth, we check to see if we can see a Barrel. If we can, we destroy it.

Method definition vs Method call

- Where are they in the slide?

```
public class Car extends Vehicle {
   public void act() {
       if(atWorldEdge()) {
           turn(15);
       }
       if(Greenfoot.getRandomNumber(100) < 20) {
           turn(Greenfoot.getRandomNumber(181) - 90);
       }
       if(Greenfoot.isKeyDown("left")) {
           turn (-36);
       }
       move();
       if(canSee(Barrel.class)) {
           destroy(Barrel.class);
       }
   }
}
```

Method Definition

```
public class Car extends Vehicle {
   public void act() {
       if(atWorldEdge()) {
           turn(15);
       }
       if(Greenfoot.getRandomNumber(100) < 20) {
           turn(Greenfoot.getRandomNumber(181) - 90);
       }
       if(Greenfoot.isKeyDown("left")) {
           turn (-36);
       }
       move();
       if(canSee(Barrel.class)) {
           destroy(Barrel.class);
       }
   }
}
```

Method calls

Where do the methods come from?

- Ourselves (the class itself can have its own methods)
- Our parent class (superclass)
- A superclass of our superclass (continue up the hierarchy)
- An outside class

How can we tell what is from where?

- If it is a method inside the same class, there will be a method definition for it in the class body.
- If it belongs to our parent or another superclass in the hierarchy, we can find out by looking at the documentation for the superclasses or in the pull-out method menus from inside the world.
- If its from an outside class, we will need to specify the class' name with a . in front of the name of the method

```
public class Car extends Vehicle {
   public void act() {
       if(atWorldEdge()) {
           turn(15);
       }
       if(Greenfoot.getRandomNumber(100) < 20) {
           turn(Greenfoot.getRandomNumber(181) - 90);
       }
       if(Greenfoot.isKeyDown("left")) {
           turn (-36);
       }
       move();
       if(canSee(Barrel.class)) {
           destroy(Barrel.class);
       }
   }
}
```

No methods called that internal to this class – there are no other methods in this class except act.

```
public class Car extends Vehicle {
   public void act() {
       if(atWorldEdge()) {
           turn(15);
       }
       if(Greenfoot.getRandomNumber(100) < 20) {
           turn(Greenfoot.getRandomNumber(181) - 90);
       }
       if(Greenfoot.isKeyDown("left")) {
           turn (-36);
       }
       move();
       if(canSee(Barrel.class)) {
           destroy(Barrel.class);
       }
   }
}
```

From an outside class – note the Greenfoot. before each one.

```
public class Car extends Vehicle {
   public void act() {
       if(atWorldEdge()) {
           turn(15);
       }
       if(Greenfoot.getRandomNumber(100) < 20) {
           turn(Greenfoot.getRandomNumber(181) - 90);
       }
       if(Greenfoot.isKeyDown("left")) {
           turn (-36);
       }
       move();
       if(canSee(Barrel.class)) {
           destroy(Barrel.class);
       }
   }
}
```

That means the rest of these are inherited from our superclasses.

Writing additional methods

- Probably a good idea to break up our code into methods.
- Better for organization
- Better for readability
- Helps to reduce errors

Writing additional methods

- Take each “piece” of the act method and break it into it’s own method.
- When complete, act will simply be a series of method calls.

```
public class Car extends Vehicle {
   public void act() {
        if(atWorldEdge()) {
            turn(15);
        }
        if(Greenfoot.getRandomNumber(100) < 20) {
            turn(Greenfoot.getRandomNumber(181) - 90);
        }
        if(Greenfoot.isKeyDown(“left”)) {
            turn (-36);
        }
        move();
        if(canSee(Barrel.class)) {
            destroy(Barrel.class);
        }
   }
}
```

```
public class Car extends Vehicle {
   public void act() {
        if(Greenfoot.getRandomNumber(100) < 20) {
            turn(Greenfoot.getRandomNumber(181) - 90);
        }
        if(Greenfoot.isKeyDown("left")) {
            turn (-36);
        }
        move();
        if(canSee(Barrel.class)) {
            destroy(Barrel.class);
        }
   }

   private void checkForEdges() {
        if(atWorldEdge()) {
            turn(15);
        }
   }
}
```

Important Note

- The line up top is now empty – that's where the code from the method used to be.
- Is that okay?

Well….

- If you want the method you wrote to never be called, then it is fine.
- However, if we actually want to check for edges of the world, we need to put in a method call to the new method we just wrote.

```
public class Car extends Vehicle {
   public void act() {
      checkForEdges();
      if(Greenfoot.getRandomNumber(100) < 20) {
          turn(Greenfoot.getRandomNumber(181) – 90);
      }
      if(Greenfoot.isKeyDown("left")) {
          turn (-36);
      }
      move();
      if(canSee(Barrel.class)) {
          destroy(Barrel.class);
      }
   }

   private void checkForEdges() {
       if(atWorldEdge()) {
           turn(15);
       }
   }
}
```

public vs private

- Note that in the method we just put in, it started with private not public – why?

public vs private

- If we are making a method that will only be used within the class, we keep it private, that way no one else is able to call it.
- If you make a method public, any other object can call that method*
  - OK, that's not quite true, but it will suffice for now until we learn about some other things.

```
public class Car extends Vehicle {
   public void act() {
        checkForEdges();
        randomTurn();
        if(Greenfoot.isKeyDown("left")) {
            turn (-36);
        }
        move();
        if(canSee(Barrel.class)) {
            destroy(Barrel.class);
        }
   }
   private void checkForEdges() {
        if(atWorldEdge()) {
            turn(15);
        }
   }
   private void randomTurn() {
       if(Greenfoot.getRandomNumber(100) < 20) {
            turn(Greenfoot.getRandomNumber(181) - 90);
        }
   }
}
```

```
public class Car extends Vehicle {
   public void act() {
        checkForEdges();
        randomTurn();
        checkKeyPressed();
        move();
        if(canSee(Barrel.class)) {
            destroy(Barrel.class);
        }
   }
   private void checkForEdges() {
        if(atWorldEdge()) {
            turn(15);
        }
   }
   private void randomTurn() {
       if(Greenfoot.getRandomNumber(100) < 20) {
            turn(Greenfoot.getRandomNumber(181) - 90);
        }
   }
   private void checkKeyPressed() {
        if(Greenfoot.isKeyDown("left")) {
            turn (-36);
        }
   }
}
```

```
public class Car extends Vehicle {
   public void act() {
        checkForEdges();
        randomTurn();
        checkKeyPressed();
        move();
        checkForBarrel();
   }
   private void checkForEdges() {
        if(atWorldEdge()) {
            turn(15);
        }
   }
   private void randomTurn() {
       if(Greenfoot.getRandomNumber
                        (100) < 20) {
      turn(Greenfoot.getRandomNumber
                  (181) - 90);
       }
   }
   private void checkKeyPressed() {
    if(Greenfoot.isKeyDown("left"))
    {
            turn (-36);
        }
    }
}
```

```
private void checkForBarrel() {
     if(canSee(Barrel.class)) {
        destroy(Barrel.class);
     }
}
```

So, in the end act is:

```
public void act() {
    checkForEdges();
    randomTurn();
    checkKeyPressed();
    move();
    checkForBarrel();
  }
```