CSSE 220

Collision Handling without InstanceOf

Checkout *DoubleDispatch* project from SVN

Important Hint

- For ArcadeGame, you will need to figure out how to draw various things on the screen
- Do this by having exactly 1 class that subclasses JComponent and invokes ordinary drawOn methods on all the other classes with the Graphics2D. That is, structured similarly to BallWorlds or BiggestFan.
- Do NOT do this by having a whole bunch of subclasses of JComponent, all of which are added to a frame or panel. You will get updating problems.

InstanceOf

- If you do inheritance correctly, you shouldn't need instanceOf…
  - General guideline: rather than instanceOf, make a method in the class your instancing
- If things get complicated we can use
  - Double Dispatch!

Bad Idea 1

```
// player has landed on o1
if(o1 instanceOf SpeedPowerUp) {
   // code to increase speed
}
if(o1 instanceOf LifePowerUp) {
 // code to increase life
}
```

Same Bad Idea

```
//player has landed on o1
if(o1.type().equals(“SpeedPowerUp”))
{
   //code to increase speed
}
if(o1.type().equals(“LifePowerUp”)) {
   //code to increase life
}
```

Simple Solution

```
o1.onPlayerCollision(player);

// in SpeedPowerUpClass
void onPlayerCollision(Player p) {
   // code to increase speed
}
```

I think you can solve BomberMan just with this simple solution, and a few special cases

```
abstract class GameObject {
    abstract boolean canBeMovedInto();
    abstract void onBombDamage();
    abstract void onPlayerCollision(..);
…etc
}
```

To Make The Simple Solution Work, you have to avoid arbitrary objects colliding with other objects

- That simple solution worked because we knew one of the objects was the Player
- What if we just have a big array of GameObjects, and sometimes GameObjects move into the same square with each other

Let's say you have this class …

```
public abstract class GameObject {

      public abstract void collide(GameObject m);

      public abstract void collide(Player m);

      public abstract void collide(PowerUp m);

      public abstract void collide(Monster m);
}
```

Late-Binding with Params? Uh oh…

- So this code:

  ```
  GameObject m = getCollidedObject();
  //We'll say getCollidedObject() returned a PowerUp
  this.collide(m);
  ```

- What method is called?
  - collide(GameObject powerup)
    - NOT collide(PowerUp powerup), even though the actual/instantiation type of m was PowerUp
- Late-binding only works for the implicit argument (what becomes **this –** thing to the left of the dot), it doesn't apply to parameter types.

This would work, but ew….

```
public abstract class Monster {

    public void collide(Monster m) {
        if (m instanceof Mushroom) {
            this.collide((Mushroom)m); return;
        }
        if (m instanceof Centipede) {
            this.collide((Centipede)m);return;
        }
        if (m instanceof Scorpion) {
            this.collide((Scorpion)m); return;
        }
    }
    public abstract void collide(Mushroom m);

    public abstract void collide(Centipede m);

    public abstract void collide(Scorpion m);
}
```

Ew means
Don't Do This!

Let's try Double Dispatch…

```
public abstract class GameObject {

	abstract void collide(GameObject m);

	abstract void collideWithPlayer(Player m);

	abstract void collideWithMonster(Monster m);

	abstract void collideWithPowerup(PowerUp m);
}
```

Double Dispatch

The key: You know your own type, so let's say we're in the PowerUp class, and GameObject is of type Monster:

```
public class PowerUp extends GameObject {
        public void collide(GameObject m) {
                m.collideWithPowerUp(this);
        }

        public void collideWithPlayer(Player p) {
                //do specific action
        }
}
```

This will call the collideWithPowerUp method on the Monster class.

Then in the Monster's collideWithPowerUp class, add code for what should happen when a Monster collides with a PowerUp.

See DoubleDispatch in repo

Work time

Be sure everyone is getting a chance to drive.

TEAM PROJECT