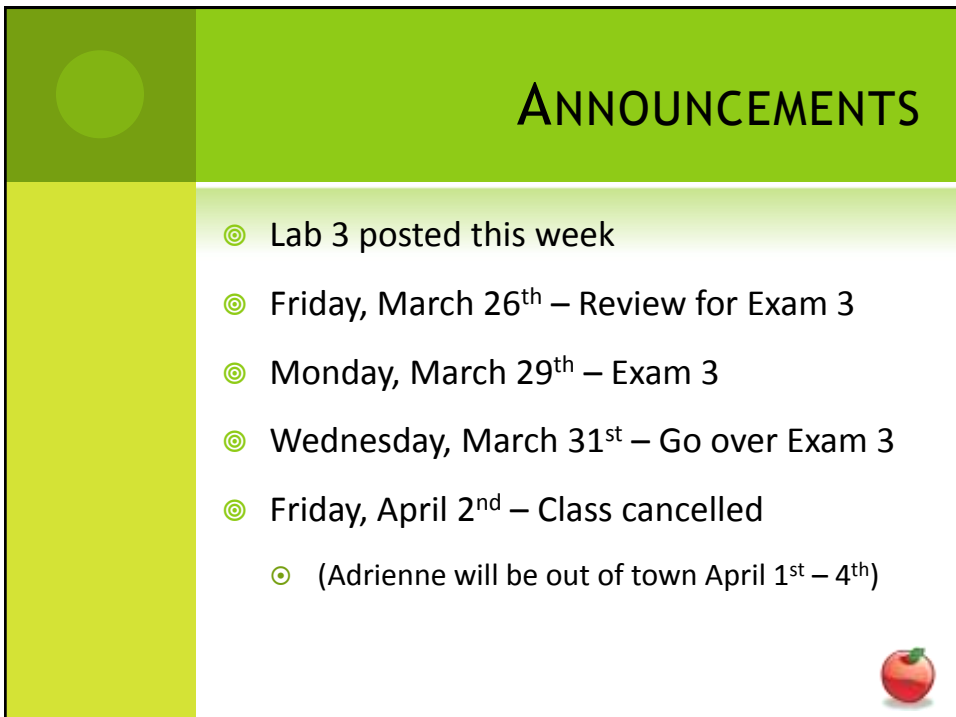



CSE 113 A

March 15-19, 2010



ANNOUNCEMENTS

- ⦿ Lab 3 posted this week
- ⦿ Friday, March 26th – Review for Exam 3
- ⦿ Monday, March 29th – Exam 3
- ⦿ Wednesday, March 31st – Go over Exam 3
- ⦿ Friday, April 2nd – Class cancelled
 - ⦿ (Adrienne will be out of town April 1st – 4th)



3

CHAPTER 6

- ⊙ Planets and gravity simulation
- ⊙ Note SmoothMover and Vector – they will be part of Lab 3 assignment as well.



4

OVERLOADING

- ⊙ Note that there are two constructors in some of the classes (like Body).
- ⊙ Normally, you would not be allowed to create two methods with the same name, but in this case it is allowed and is called method overloading.
- ⊙ Method overloading (having two methods with the same name in the same class) is only allowed when the methods differ in the number and/or type of parameters.



5

APPLY GRAVITY (TO A PARTICULAR PLANET)

- ⦿ First, get all the bodies in the scenario
- ⦿ Then, apply the force of gravity from each body so that it impacts the motion (Vector) of the “current” planet



6

GETTING THE PLANETS

- ⦿ `getWorld().getObjects(Body.class)`
 - ⦿ Returns a list that we need to store
- ⦿ `java.util.List<Body> bodies;`
 - ⦿ Creates a variable that holds onto a list of Body objects
- ⦿ `bodies = getWorld().getObjects(Body.class);`
 - ⦿ Assigns the list of bodies to the variable we've just created



7

NOW WHAT?

- ⦿ So, we have a list of planets, but now we need to cycle through the list and use each planet to help calculate movement.
- ⦿ We can use a for-each loop to cycle though (or iterate over) the list of planets.



8

FOR-EACH LOOP (SYNTAX)

```
for(NumberOfElementInCollection variableName: nameOfCollection)
{
    //what to do with each element
}
```



9

FOR THE PLANETS

```
for(Body b: bodies)
{
    applyGravity(body);
}
```



10

Get the ball moving

- create a new vector with
dx & dy other than 0
- set Movement



11

Check for Edges

```
if (hit top)
  bounce -reversing vertical direction
else if (hit Bottom)
  stop scenario
if (hit left)
  bounce
else if (hit right)
  bounce
```



12

Check for paddle

```
if (collided with paddle)
  bounce off paddle
```



13

OPTIONAL

- ⊙ Making the ball change its angle when it hits the paddle.
 - ⊙ Reverse the y-direction the ball was moving
 - ⊙ Find the center of the ball
 - ⊙ Find the center of the paddle
 - ⊙ If the difference between them is zero, do nothing.
 - ⊙ Otherwise, set the dx of the ball to be
 - center X of ball – center X of paddle



14

Check for Bricks

if (collided with brick)

- remove brick
- bounce the ball



15

Moving the paddle

Mouse Info `m = Greenfoot.getMouseInfo();`

↑
`m.getX()`

returns the x-location
of the mouse

We can set our paddle's
x-location to that
spot.



16

When you break a brick, 70% of
the time, nothing special happens,
but 30% of the time
random Activity is called.



17

Random Activity chooses between
the five activities

```

if (randomNumber == 0)
    slowDownBall();
if (randomNumber == 1)
    speedUpBall();
if (randomNumber == 2)
    moveBrickDown();

```



18

Stuff w/ Bricks

- get list of bricks
- use for-each loop to do something to each brick
 - set location of each brick
 - set the color to be BLUE
 - set the color to be random



19

JAVA.AWT.COLOR

- ⦿ There are several pre-defined colors in Java that you can use
- ⦿ `java.awt.Color.PINK`, `java.awt.Color.RED`,
`java.awt.Color.ORANGE`, `java.awt.Color.YELLOW`,
`java.awt.Color.GREEN`, `java.awt.Color.CYAN`,
`java.awt.Color.BLUE`, `java.awt.Color.MAGENTA`,
`java.awt.Color.LIGHT_GRAY`, `java.awt.Color.GRAY`,
`java.awt.Color.DARK_GRAY`, `java.awt.Color.BLACK`,
`java.awt.Color.WHITE`



20

JAVA.AWT.COLOR

- ⦿ You can also create a color using
`new java.awt.Color(red, green, blue)`
- ⦿ where you substitute a number within the range 0-255 for each of red, green, and blue

