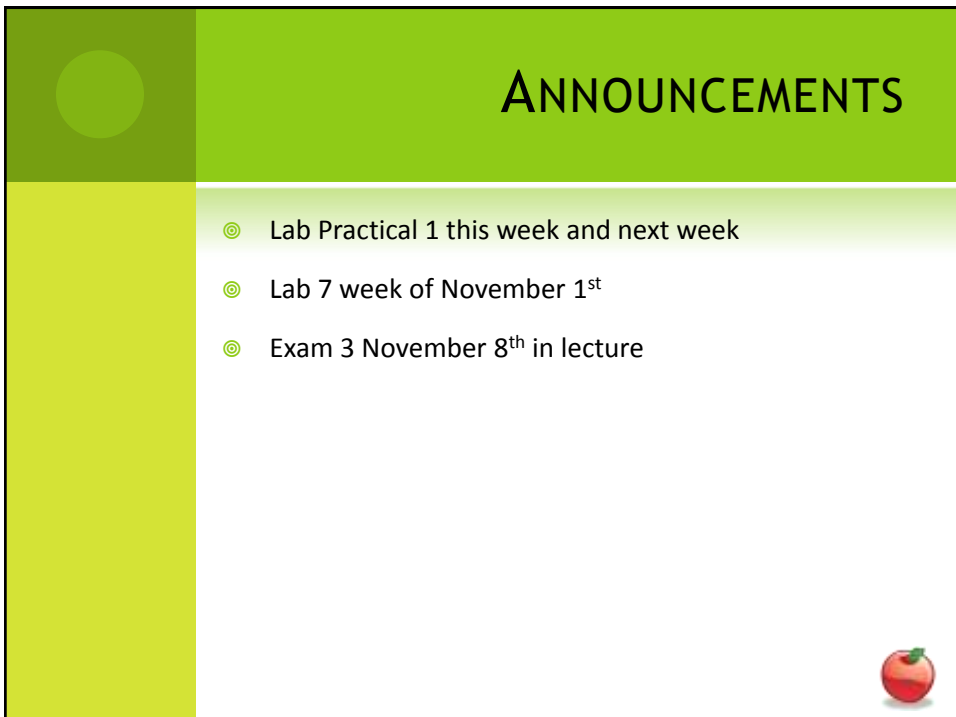



CSE 113 B
October 18 – 22, 2010



ANNOUNCEMENTS

- ⊙ Lab Practical 1 this week and next week
- ⊙ Lab 7 week of November 1st
- ⊙ Exam 3 November 8th in lecture



3

MOVEMENT WITH VECTORS

- ⦿ Review movement with vectors
- ⦿ Turning randomly by changing rotation of graphic and changing the direction in the vector



4

OVERLOADING

- ⦿ Note that there are two constructors in some of the classes (like SmoothMover).
- ⦿ 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

COLLIDING OBJECTS

- ⦿ Detecting intersecting objects can be done using `getOneIntersectingObject` method.
- ⦿ This method can take as an argument a class that represents the type of object we are looking for (like `canSee` in Crab example).
- ⦿ This method returns an Actor object that represents what the current actor is intersecting with. If there is no intersecting actor, the method returns `null`. `null` is a keyword in Java that represents the value of a null reference (can be thought of as “no object”).



6

CASTING

- ⦿ Recall from earlier examples the following code:

```
Actor a = getOneIntersectingObject(X.class);
```
- ⦿ Remember that X is the class we are interested in looking for collisions with – it can be anything (Flower, Ball, Brick, Barrel).
- ⦿ `getOneIntersectingObject` returns the object we are interested with or `null` if not intersecting an object of the passed-in type. The object that is passed back is of type Actor.



7

CASTING

- ⊙ Therefore, the type of the variable a is Actor.
- ⊙ If we try to do this:

```
X a = getOneIntersectingObject(X.class);
```

- The code will not compile because `getOneIntersectingObject` returns an Actor, not an X.
- But we know that the Actor that is really being returned is an X.



8

CASTING

- ⊙ However, sometimes we may want to do things with a (the variable) that only X's can do.
- ⊙ However, a is an Actor and can only do things Actors can do.
- ⊙ If we want to treat the object that is returned by `getOneIntersectingObject` as an X, we can explicitly cast it as an X.



9

CASTING

```
X a = (X) getOneIntersectingObject(X.class);
```

- ⦿ The (X) is the cast.



10

GETTING ALL THE BARRELS

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



11

NOW WHAT?

- ⦿ So, we have a list of barrels, but now we need to cycle through the list and move each of them down on the screen.
- ⦿ We can use a for-each loop to cycle through (or iterate over) the list of barrels.



12

FOR-EACH LOOP (SYNTAX)

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



13

FOR THE BARRELS

```
for(Barrel b: barrels)
{
    setLocation(getX() + 10, getY());
}
```



14

REMOVE FROM WORLD

- ⦿ Note that there is a method for removing all objects of a specific type from a world (see World's documentation).

