CSE 113 A

February 7 – 11, 2011

Announcements

- Exam 1 Review in lecture 2/11
- Exam 1 in lecture 2/14
- Lab 1 due 2/18
- Practical Exam 1 in recitation week of 2/21
- This week in recitation: Lab 3 & Practice Assignment 3

Recap Slides (Last Week's Material)

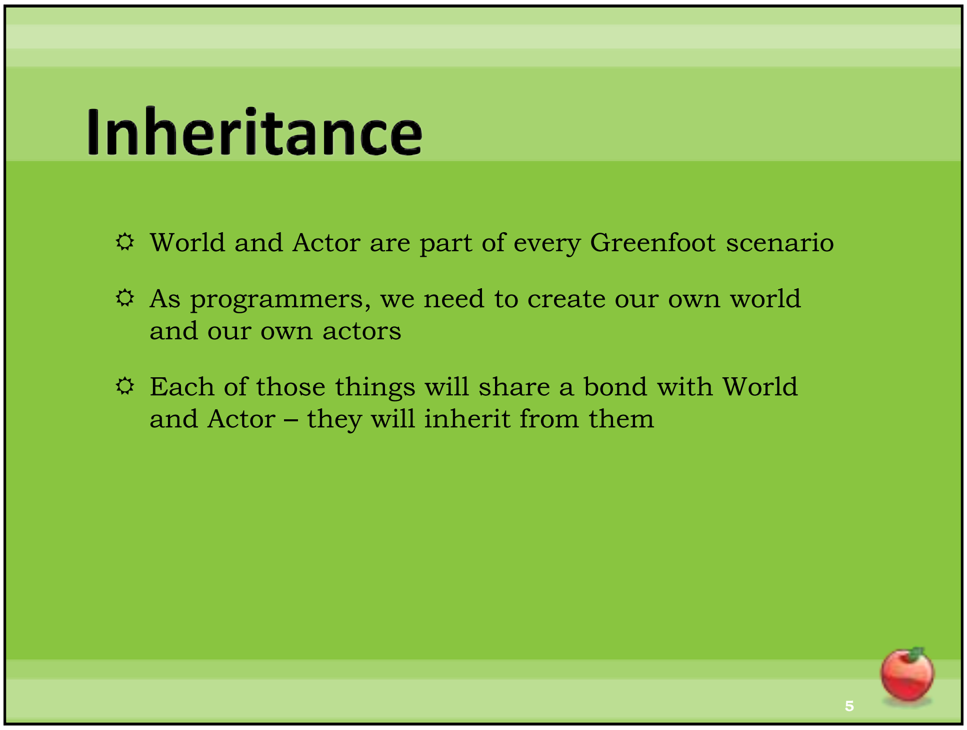
- Greenfoot Environment
  - Class diagram panel
  - Execution Controls
  - World
- Creating an object in the world
  - Right clicking brought up method menu

3

Parameters

- When we describe a method to someone, like what is listed in the right-click menu in Greenfoot, we give the parameter's type
- When we execute that method, we need to an actual value to that parameter
  - Example from lecture – set direction method – we needed to say which direction 0,1,2, or 3

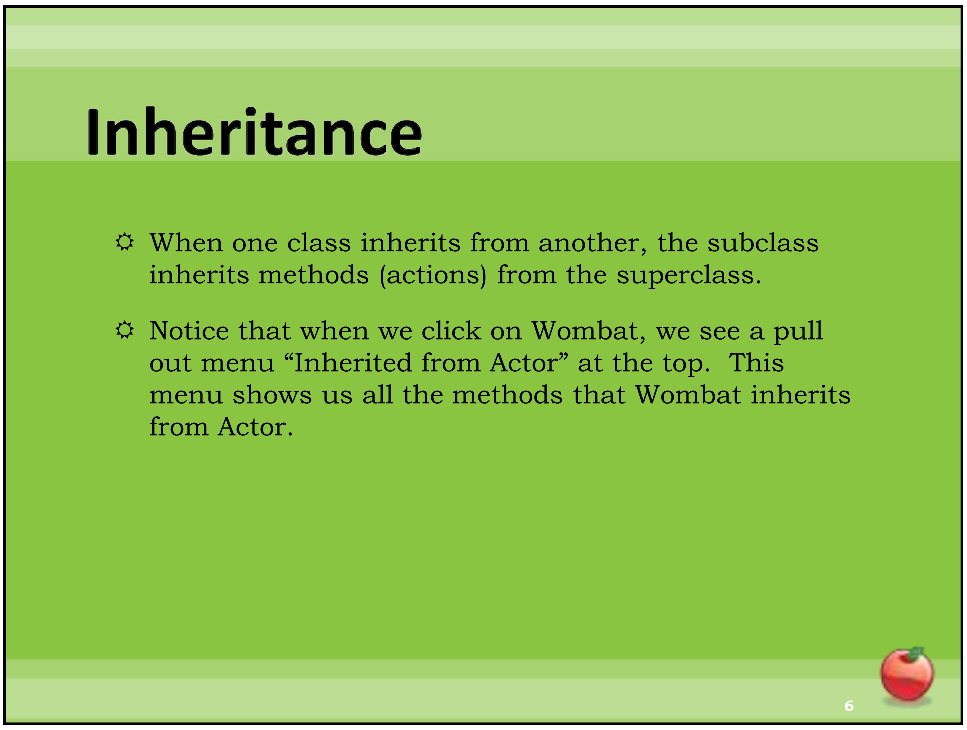
4



Inheritance

- World and Actor are part of every Greenfoot scenario
- As programmers, we need to create our own world and our own actors
- Each of those things will share a bond with World and Actor – they will inherit from them

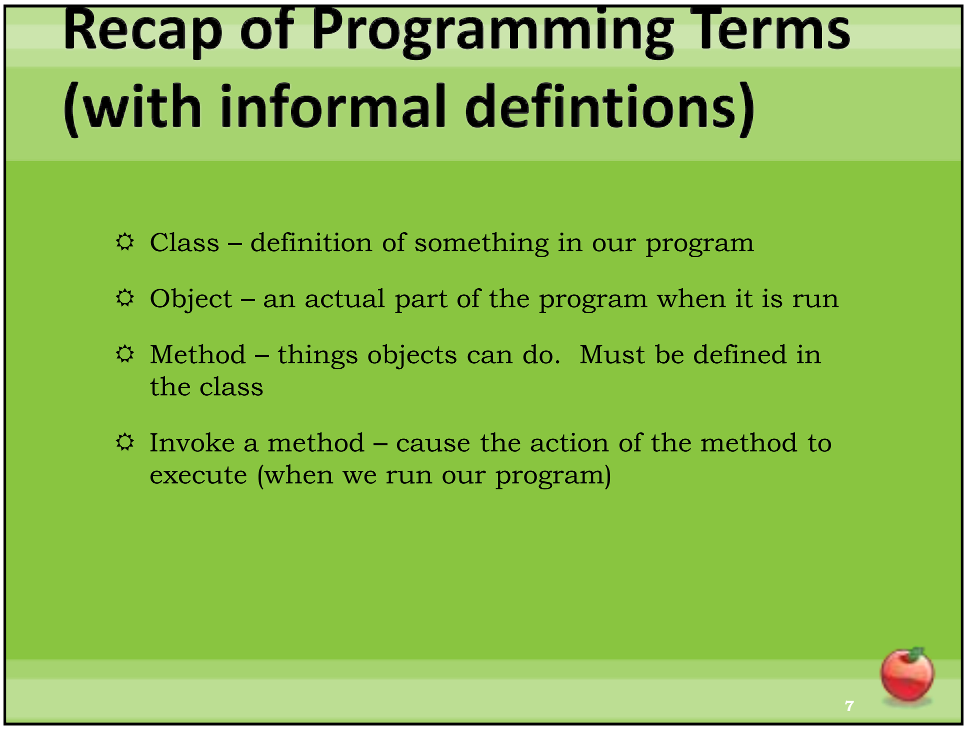
5



Inheritance

- When one class inherits from another, the subclass inherits methods (actions) from the superclass.
- Notice that when we click on Wombat, we see a pull out menu “Inherited from Actor” at the top. This menu shows us all the methods that Wombat inherits from Actor.

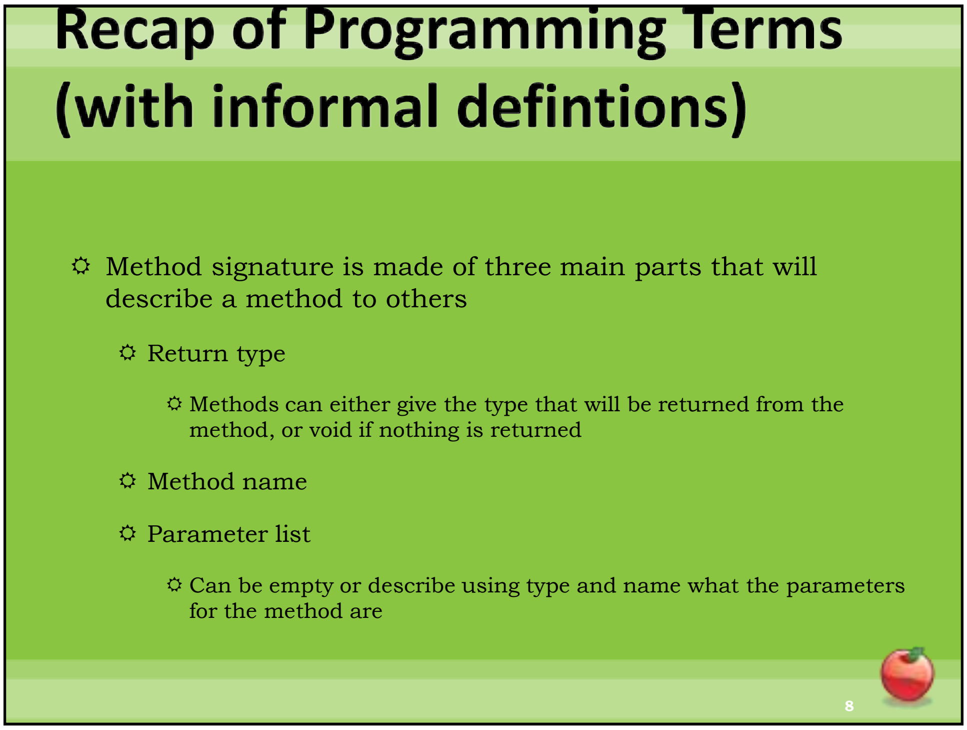
6



Recap of Programming Terms (with informal defintions)

- Class – definition of something in our program
- Object – an actual part of the program when it is run
- Method – things objects can do. Must be defined in the class
- Invoke a method – cause the action of the method to execute (when we run our program)

7



Recap of Programming Terms (with informal defintions)

- Method signature is made of three main parts that will describe a method to others
  - Return type
    - Methods can either give the type that will be returned from the method, or void if nothing is returned
  - Method name
  - Parameter list
    - Can be empty or describe using type and name what the parameters for the method are

8

More Ideas

- "Running" a scenario in Greenfoot causes the act methods of all the actors in the scenario to be invoked repeatedly
- All Greenfoot scenarios use inheritance. World and Actor are superclasses of the specific type of world and specific type of actors the programmer creates for the scenarios. The specific types (subclasses) inherit the methods from the more general types (superclasses)

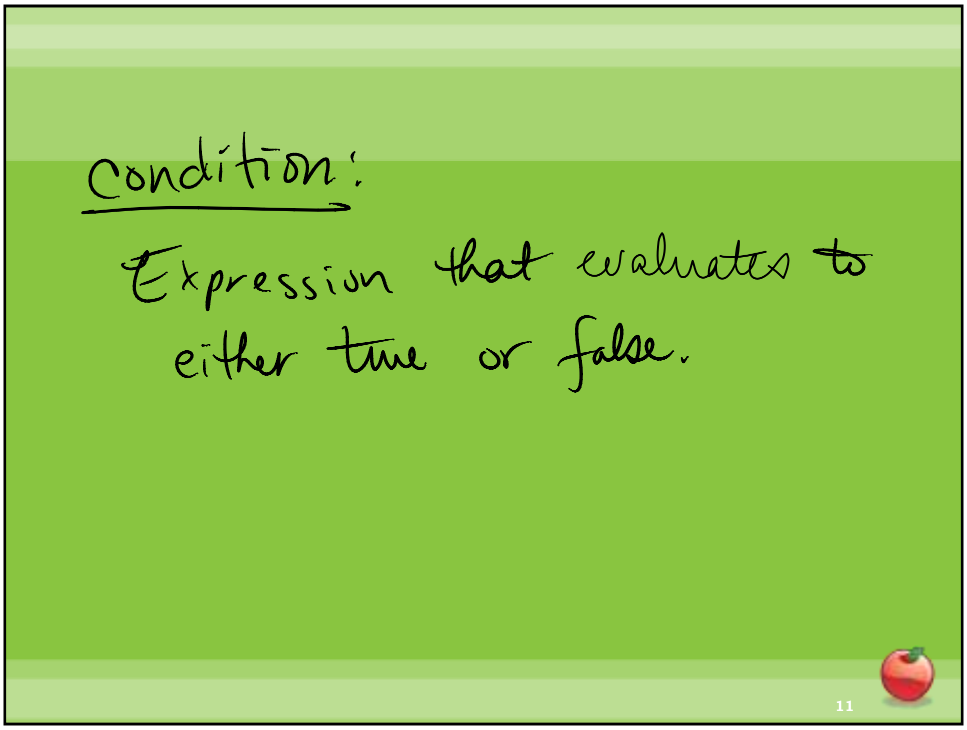
9

If-statements (Selection)

Syntax:

```
if ( condition )
{
   code that is executed if the
   condition is true
}
```

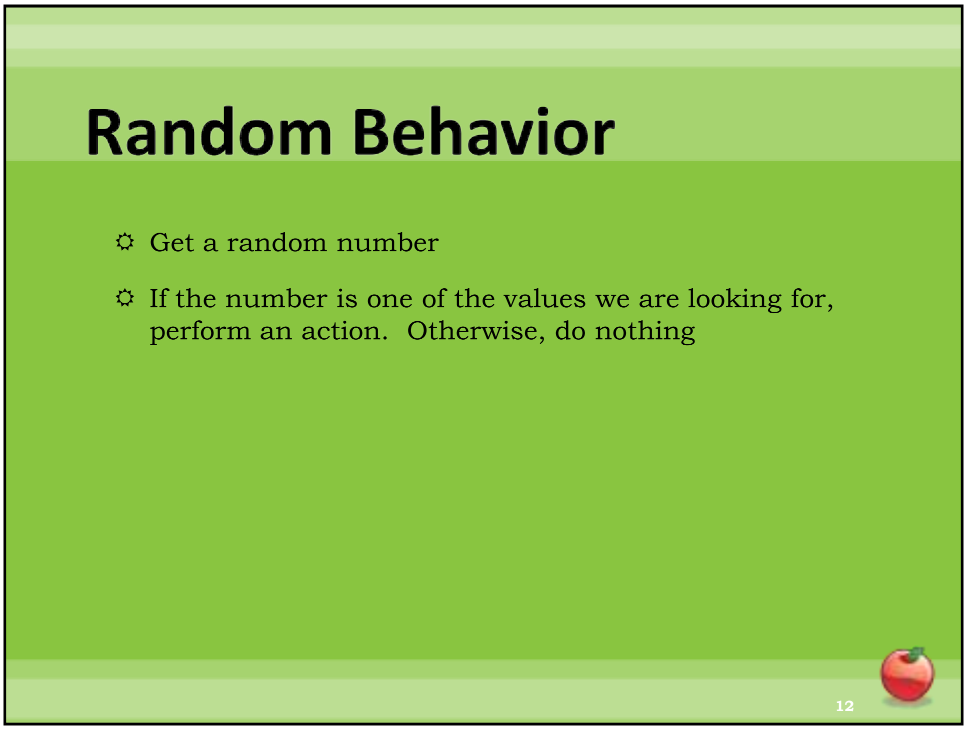
10



condition:

Expression that evaluates to either true or false.

11



Random Behavior

- Get a random number
- If the number is one of the values we are looking for, perform an action. Otherwise, do nothing

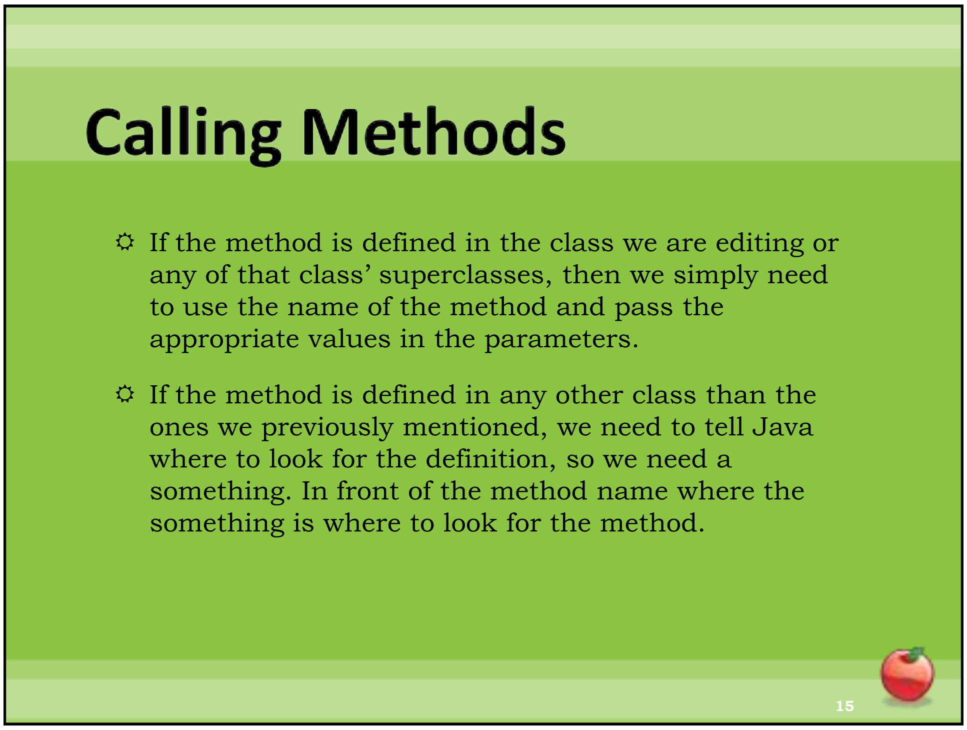
12

Getting a Random Number

- Greenfoot provides a special method for getting random numbers. To call it, we need to use the following:
- `Greenfoot.getRandomNumber(x);`

Greenfoot. ?

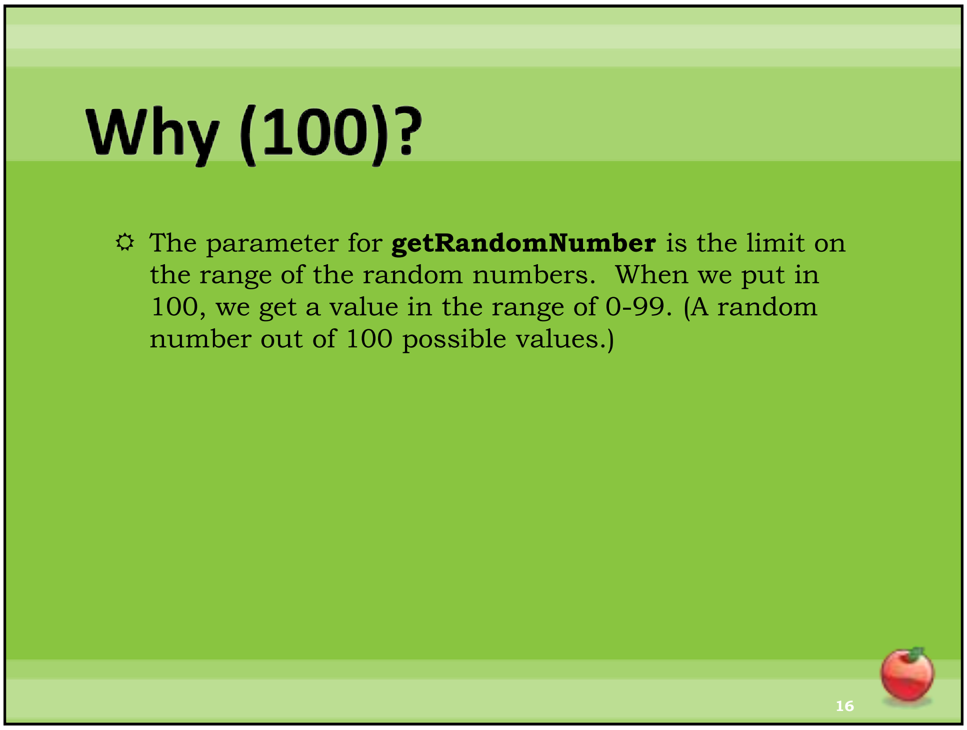
- The name of this method is **getRandomNumber**, but there is a **Greenfoot.** in front of that method name.
- **getRandomNumber** is a method that is defined inside of a class named **Greenfoot**, so we need to tell Java where to find this method – hence the **Greenfoot.**
- Without the **Greenfoot.** in front, Java looks inside the current class and inside the superclasses for a definition for that method. Since there isn't one, it would cause a compiler error if we left off the **Greenfoot.** In this example.



Calling Methods

- If the method is defined in the class we are editing or any of that class' superclasses, then we simply need to use the name of the method and pass the appropriate values in the parameters.
- If the method is defined in any other class than the ones we previously mentioned, we need to tell Java where to look for the definition, so we need a something. In front of the method name where the something is where to look for the method.

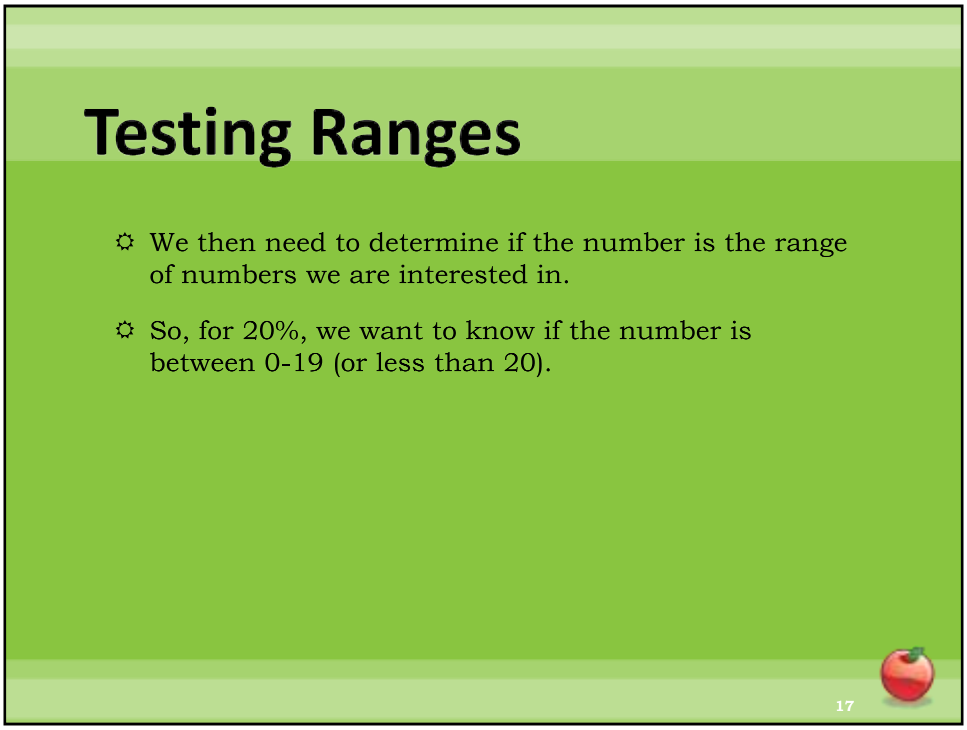
15



Why (100)?

- The parameter for **getRandomNumber** is the limit on the range of the random numbers. When we put in 100, we get a value in the range of 0-99. (A random number out of 100 possible values.)

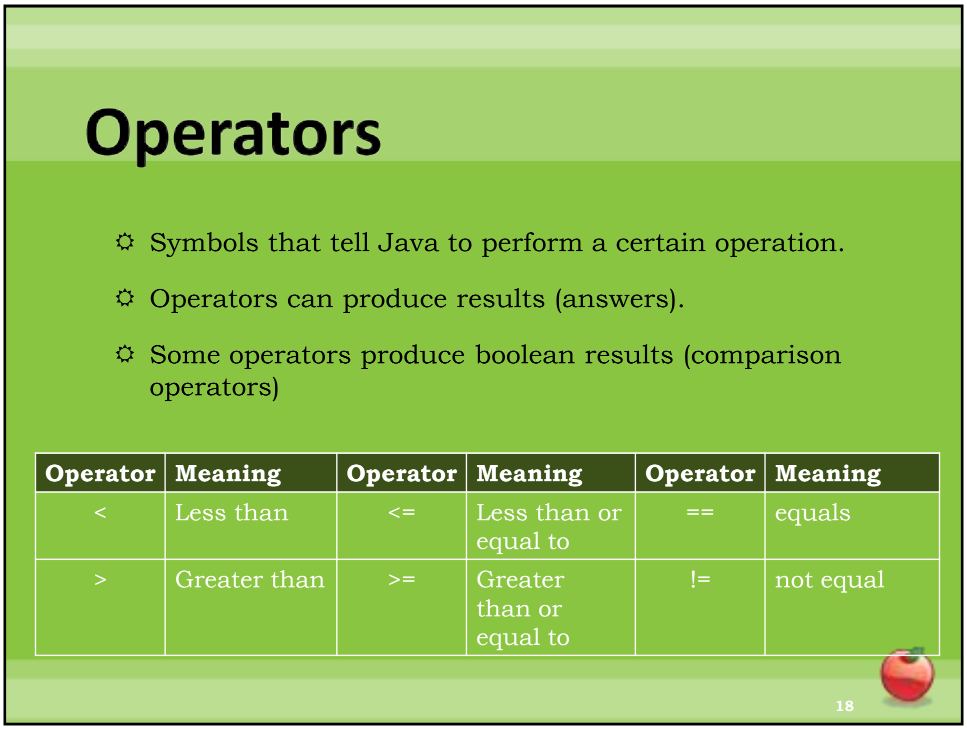
16



Testing Ranges

- We then need to determine if the number is the range of numbers we are interested in.
- So, for 20%, we want to know if the number is between 0-19 (or less than 20).

17



Operators

- Symbols that tell Java to perform a certain operation.
- Operators can produce results (answers).
- Some operators produce boolean results (comparison operators)

Operator	Meaning	Operator	Meaning	Operator	Meaning
<	Less than	<=	Less than or equal to	==	equals
>	Greater than	>=	Greater than or equal to	!=	not equal

18