**JavaTown Objects Reading**

x = 1; //create a new variable x and assign it the value of 1

x = 2; //redefine the variable x to the value 2

x = 3; //redefine the variable x to the value 3

x = 4; //type me too! //redefine the variable x to the value of 4 and demonstrate that JavaTown simply ignores anything after the //

x = 3; //x = 4; //revert the variable x back to value 3 and show that JavaTown ignores code that is commented out, just like Java

x = x + 7; //Redefine x to be its current value + 7, or 10.

y = 9 / 4; //Division in Java gives the integer part of the quotient, so this would define a variable y with value 2. The % operator, on the other hand, gives the remainder.

y = 2 \* (3 + 4); //Reassigns y to the right-hand expression, which turns out as 14

y=8%3; //returns the remainder when 8 is divided by 3

x = 3 > 1; //x is now a Boolean that is true (because 3 is greater than 1)

x = 3 <= 1; //x is now false, because 3 is not less than or equal to 1. Note: In actual Java, one does not have the freedom to switch between types of primitives this easily.

iLoveComputers = true; //make a new variable iLoveComputers and assign it the value true

iLOVEComputers = false; // make a new variable iLOVEComputers and assign it the value false (note the difference in capitalization)

x = !(3 > 2); //x is still a Boolean that represents whether 3 is not greater than two; it will return false (the ! means not)

x = 1 == 2; //x returns false, because 1 does not equal 2

x = 1 != 2; //x returns true, because 1 does not equal 2

x = 1==1 && 1==2; //x returns false, because both the conditions must be met; 1 is not equal to 2, so both conditions are not met

x = 3/0 == 3; //Run-time error: division by 0 prevents it from executing anything

x = 1==1 && 3/0==3; //Run-time error because of dividing by 0; it doesn’t care that 1==1 is true, it cares about BOTH being true

x = 1==1 || 3/0==3; //This will return true because ONE is true; Java doesn’t care about the second thing in the statement.

x = false || 3/0==3; //This will return a run-time error because x isn’t false; it’s true, and 3/0 does not equal 3, so it will return a error

x = false && 3/0==3; //This will return false, because x is true AND 3/0 doesn’t equal 3

x = nutella; //This is an unbound variable, and it will return an error as such.

x = null; //x doesn’t refer to anything.

x = "Nutella";//x is now a string with the value “Nutella”.

count = 3; // new variable count with value 3

count = count + 1; //redefine count to be count + 1, or 3 + 1 = 4.

if (5 > 3) result = "yay"; // define a new variable result with the value “yay” if 5 >3; it will have this value because 5 IS greater than 3

if (false) result = "boo"; // tests if something is false; if it’s not, nothing happens – like in this case

if ("false") result = "huh?"; // a string cannot be cast to a boolean – it gives an error (unknown error)

if (5 == 3) // tests if five is equal to three

status = "bad"; //if the condition is true, define variable status as

“bad”

else

status = "good"; //if the condition is false, define status as “good”

if (1 + 2 == 3) // test if 1 + 2 is equal to 3

{

result = "bad"; // if that is true then set result to “bad”

result = "i mean good"; // change it to “I mean good”

}

public class Noob {} // define a new class Noob

n1 = new Noob(); //initialize a new variable n1 referring to an object of

class Noob()

n2 = n1; //this creates a new reference to the Noob() referred to by reference n1, the object is still the same though

same = (n1==n2); // this is true because n1 and n2 refer to the same object Noob() on the heap

n2 = new Noob(); //this makes a new Noob() with n2 referring to it

same = (n1==n2); // now this returns false because n1 and n2 refer to different Noob() objects on the heap

n2 = null; //n2 doesn’t refer to anything, and the object on the heap has no reference and is useless.

public class Noob // redefine the same class Noob

{

public speak() { return "lol"; } //give it a speak method to return “lol”

}

speech = n1.speak(); //the Noob (referenced by n1) says “lol”

speech2 = n2.speak(); //the Noob (referenced by n2) says “lol”

public class Noob //redefining the class Noob again.

{

public speak() { return "lol"; } //same speak method

public askFriendToSpeak(friend) // new askFriendToSpeak method that calls the argument friend (i.e. the reference to another Noob)

{

spoken = friend.speak(); // assign the variable spoken to whatever the friend had to say (aka “lol”)

return "friend said " + spoken; // returns “friend said lol”

} // close method

} // close class

n2 = new Noob(); //we create a new Noob and refer to it with n2

speech = n2.askFriendToSpeak(n1); //we call this new method on n2 and pass it the argument n1, and it will return “n1 said lol” in a variable speech

public class Lamp // new class Lamp

{

private lightOn; //private instance variable lightOn (boolean)

public turnLightOn() { lightOn = true; } //method turnLightOn switches the state of lightOn to true

public turnLightOff() { lightOn = false; }//method turnLightOff switches the state of lightOn to false

}// close class

public class Lamp // new class Lamp

{

private lightOn; //private instance variable lightOn (boolean)

public Lamp(init) { lightOn = init; } // the constructor now takes an argument init that will be assigned to the instance variable lightOn

public turnLightOn() { lightOn = true; } //method turnLightOn switches the state of lightOn to true

public turnLightOff() { lightOn = false; }//method turnLightOff switches the state of lightOn to false

}// close class