CECS 174 – Lecture 23 – Objects and Classes

Class - A class is a programmer defined data type that allows you break up your program into more manageable pieces. They are made up of instance variables that store information about the object (the things an object has), and methods to access or alter the object's data (the things an object does). A class acts as a sort of template to create one or more objects.

Object - An object is an instance of a class. Objects are created in the main section of your program and are manipulated by using the object's methods, which are defined inside of the class. These methods can access and change the object's instance variables.

Example:

The Rectangle class has four instance variables: an x and y position of the upper left hand corner, a width, and a height. Then it has several methods that allow the rectangle to be manipulated (only three are listed here, but there are many others in the Rectangle Class).

Rectangle
int x int y int width int height
void setRect(int x, int y, int width, int height) void translate(int x, int y) int getX(), int getY() int getWidth(), int getHeight()

Creating an Object

To create a new object, it needs to be constructed and given a set of initial parameters. Most objects can be programmed to use a set of default values, but a constructor still needs to be run first to create the object and set up the default values.

```
className variableName = new className(parameters);
```

The first className is defining the type that will be assigned to your variable. Then the constructor is assigning a new instance of the object with the given parameter values to the object's variables.

Example:

```
Rectangle box1 = new Rectangle( 0, 0, 10, 5 );
```

An instance of the class Rectangle is being assigned to the object variable box1 with the type Rectangle. The constructor sets the object's (box1) x and y location to (0, 0), its width to 10, and its height to 5.

Using Objects

Once an object is created, we can use it within our program. However, most of the time an object's instance variables are protected and are not directly accessible, so we need to use the object's methods to access them.

```
//resets the rectangle's location and size
box1.setRect( 0, 0, 10, 5 );

box1.translate( 3, 4 ); //moves box right 3 down 4

//get's the rectangles location and displays it
System.out.println("X = " + box1.getX());
System.out.println("Y = " + box1.getY());
```

Object References

The variable `box1`, does not contain the actual data of the object, it only contains a reference (location in memory) to the object. If a copy of box1 is made (box2 = box1), it will not create a new copy of the Rectangle, it will create a copy of the reference to the Rectangle. In other words, they will both be pointing to the same memory location. This means that if you alter the Rectangle using box1, it will alter the same Rectangle that box2 is pointing to as well.

Example:

```
Rectangle box2 = box1;
box2.setRect( 10, 10, 20, 15 );
```

Not only has box2 been reset, but if box1 were displayed we would see that box1 has been reset to the new values as well.
If you want box2 to be its own Rectangle then the new operator must be used to construct a new Rectangle for box2.

```
Rectangle box2 = new Rectangle (10, 10, 20, 15 );
```

Example:

```
Rectangle box1 = new Rectangle ( 50, 20, 10, 20 );
Rectangle box2 = new Rectangle ( 25, 25, 5, 15 );
Rectangle box3 = box1;
box1.translate( 10, 15 );//modifies x and y values
box2.translate( 5, 10 );
box3.translate( 5, 5 );
```

box1: x=_____, y= _____, height=_____, width=_____

box2: x=_____, y= _____, height=_____, width=_____

box3: x=_____, y= _____, height=_____, width=_____

Using an Array of Objects

Creating an array of objects is very similar to creating any other type of array except that the data type is the class name that you are using.

Example – Creating an array of 10 Rectangles

```
Rectangle [] boxes = new Rectangle[10];
```

The only difference is now each of the objects needs to be constructed separately.

```
for( int i=0; i < boxes.length; i++){
     boxes[i] = new Rectangle(0,0,0,0);
}
```

The object's methods can still be accessed as needed by specifying an index.

```
for( int i=0; i < boxes.length; i++){
     System.out.print(boxes[i].getX()+ " ");
     System.out.println(boxes[i].getY());
}
```

Using Objects in Methods

Objects can also be used as parameters or return values in methods in a similar way to what we are used to.

Example 1 – passing a Rectangle as a parameter (passed by reference, like an array)

```
public static void printRect( Rectangle r ){
     for( int i=0; i< r.getHeight(); i++){
          for(int j=0; j< r.getWidth(); j++){
               System.out.print("* ");
          }
          System.out.println();
     }
}
```

Example 2 - using a Rectangle as a return value

```
public static void main(String[] args){
     Rectangle box = makeSquare();
}
public static Rectangle makeSquare( ){
     System.out.print("Enter an x and y:");
     int x = in.nextInt();
     int y = in.nextInt();
     Rectangle r = new Rectangle(x, y, 5, 5);
     return r;
}
```