Class – is the blueprint/template of an object. It also allows us to define advanced datatypes for situations which cannot be be met by primitive datatypes.

Classes have:

1. State – Represented by member fields/variables
2. Behaviour – Represented by methods

**Encapsulation** – declare private variables. This means hiding the variables/fields and methods from outside world. Internal representation of an object is hidden from view outside of object definition. So internal of the object is only accessible by this class.

Encapsulation allows you to do some validations on the input parameters sent to setter(), like age cannot be -ve. Because if you make everything public then you let you class to be accessed and handled in undesirable ways. It also makes a client of a class to be tightly couple with the class. Like if you change the name of public class. It may also create problem with object creation and initialization.

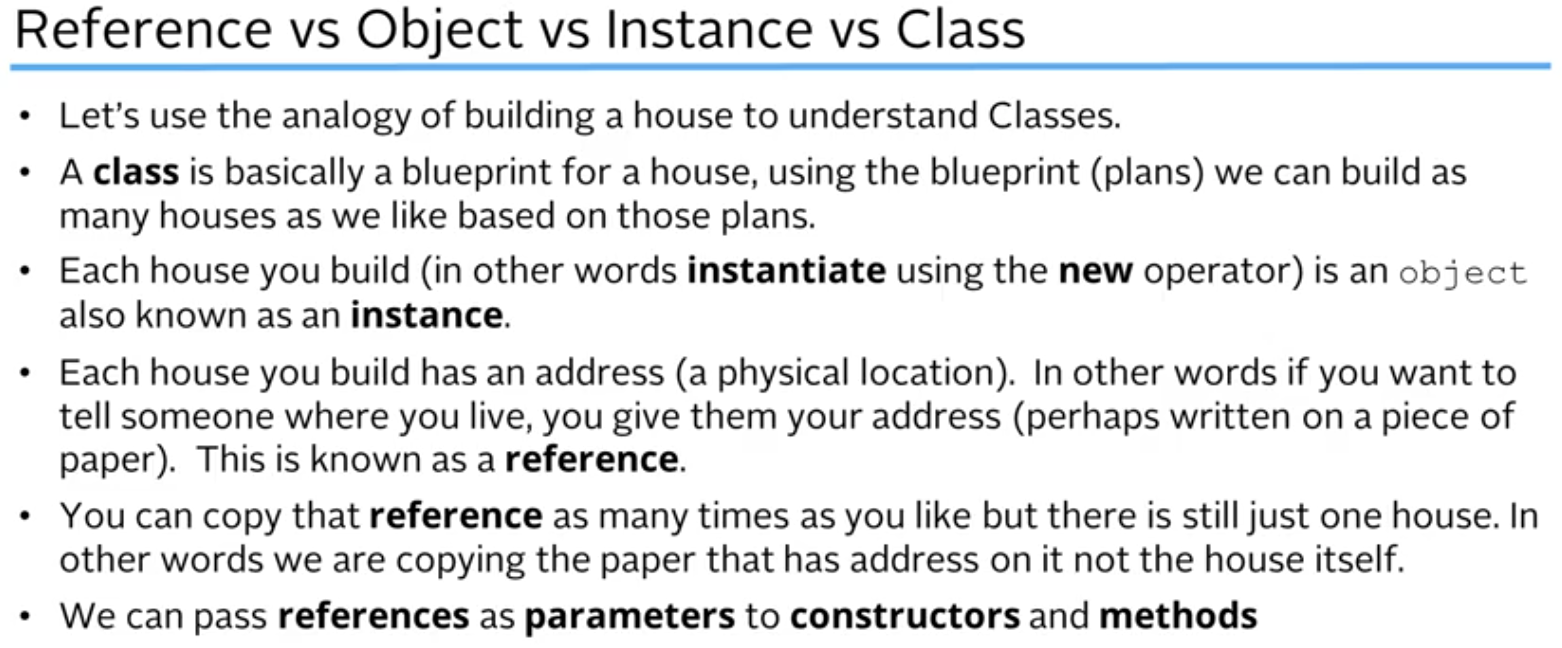
You can call one constructor from another constructor, and that is done as the very first line in the calling constructor.

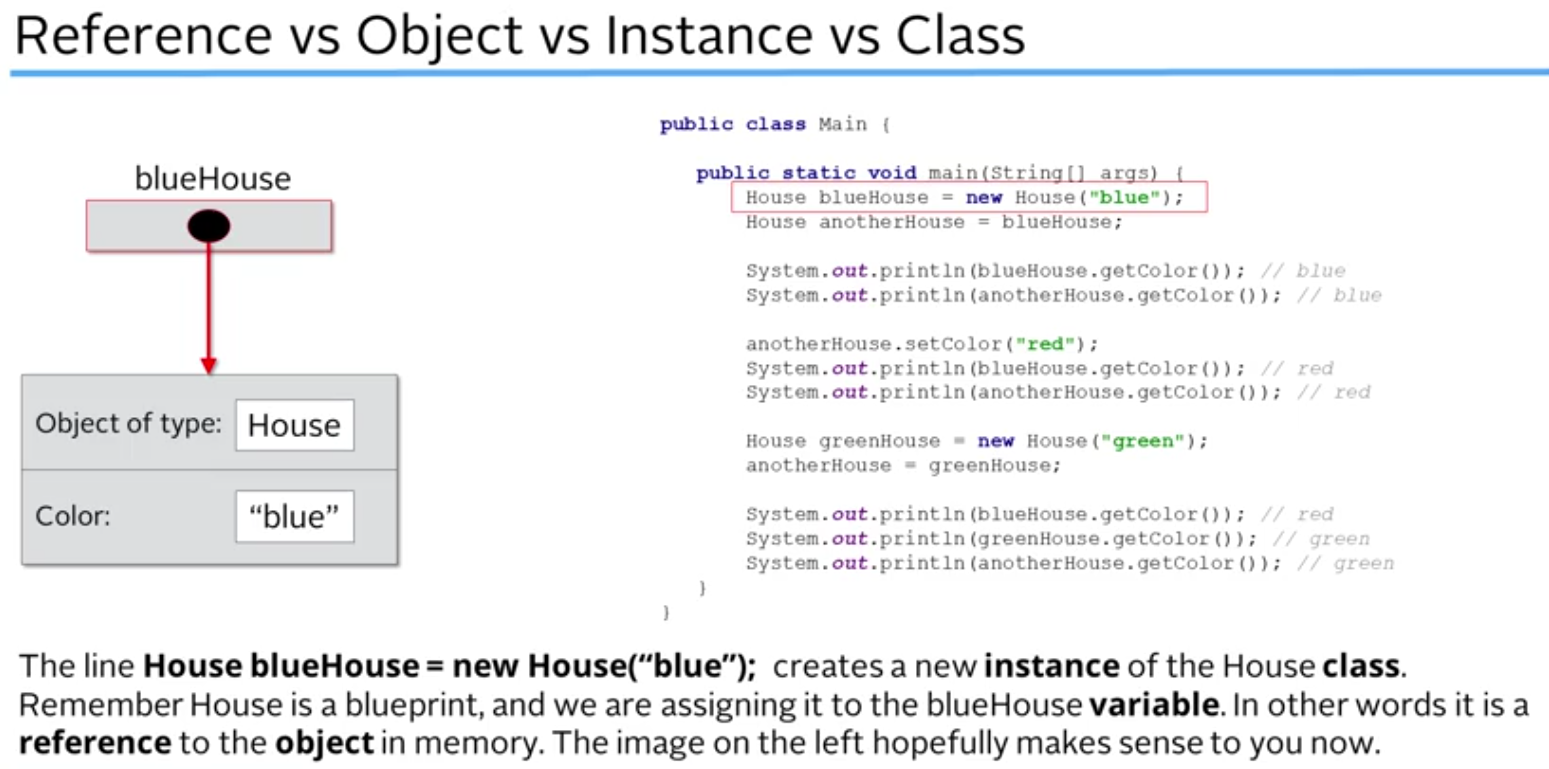
A general rule is that you do not call a setter method in a constructor. You generally populate he member variable using this.var = var. One reason may be that some other initialization might not have been over which may be needed for that setter.

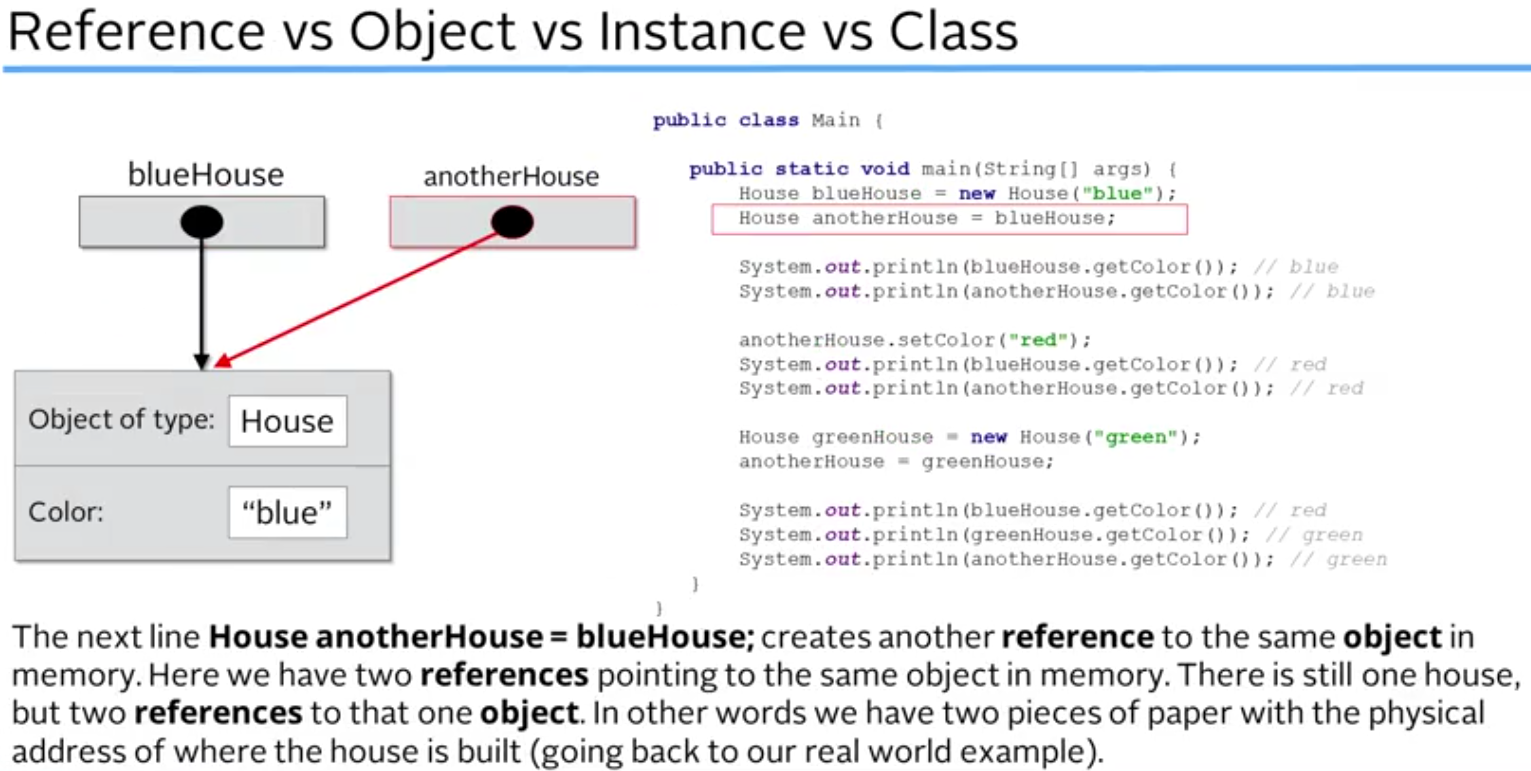
You may sometimes want to call the setter method if you have some validation being performed in the setter method.

If the parent class has a no-arg constructor then the base class should have a constructor which calls the super class constructor using super(…).

It is better to call the method of the super class without the keyword super because if you override the super class method then using super.method() will not call your overridden method. But calling method() will call your overridden method. Calling super.method() does not look into the current (child class method()). It directly calls the base class method(). Whereas if you only call method() then it will look for the method() implementation in current class.

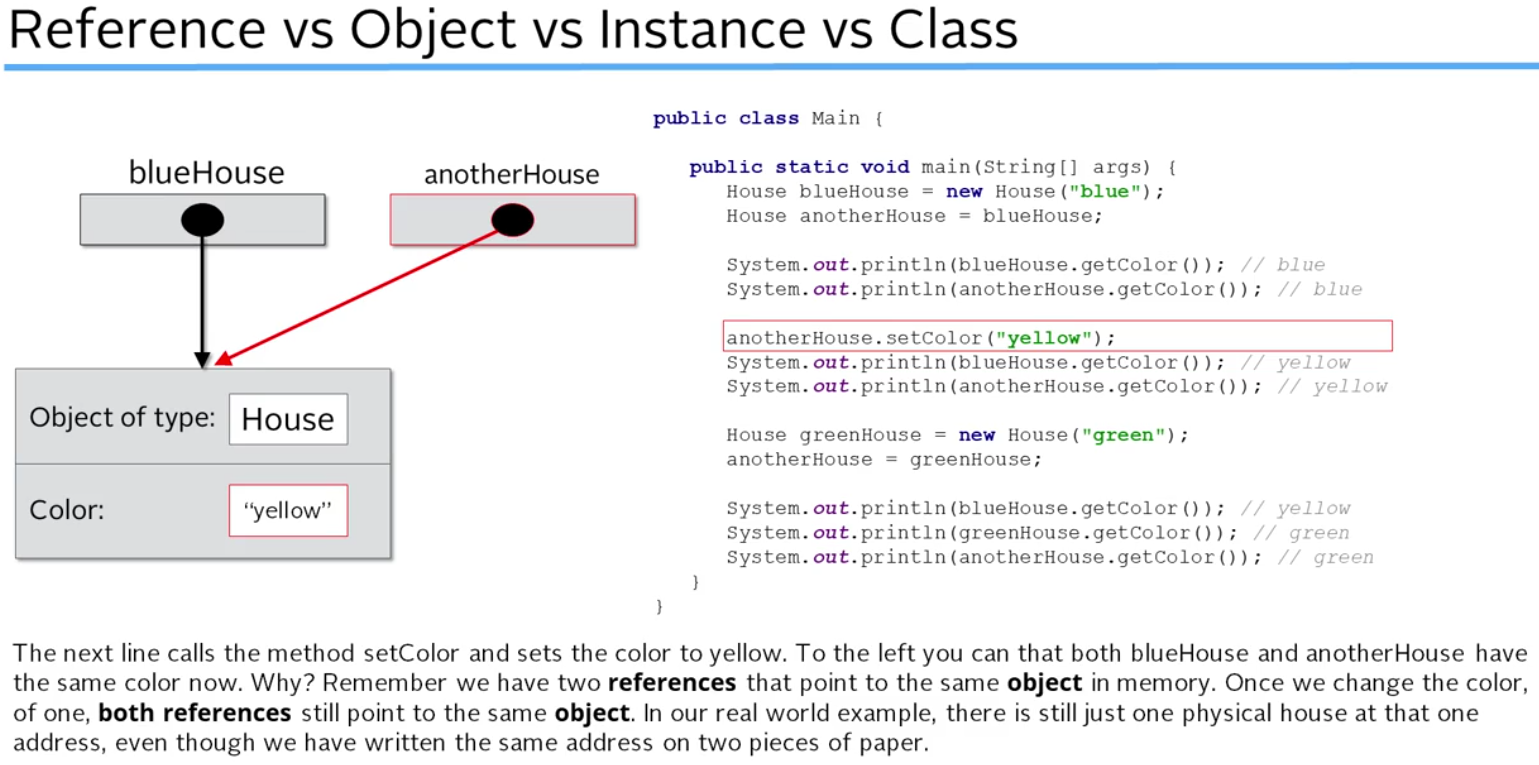


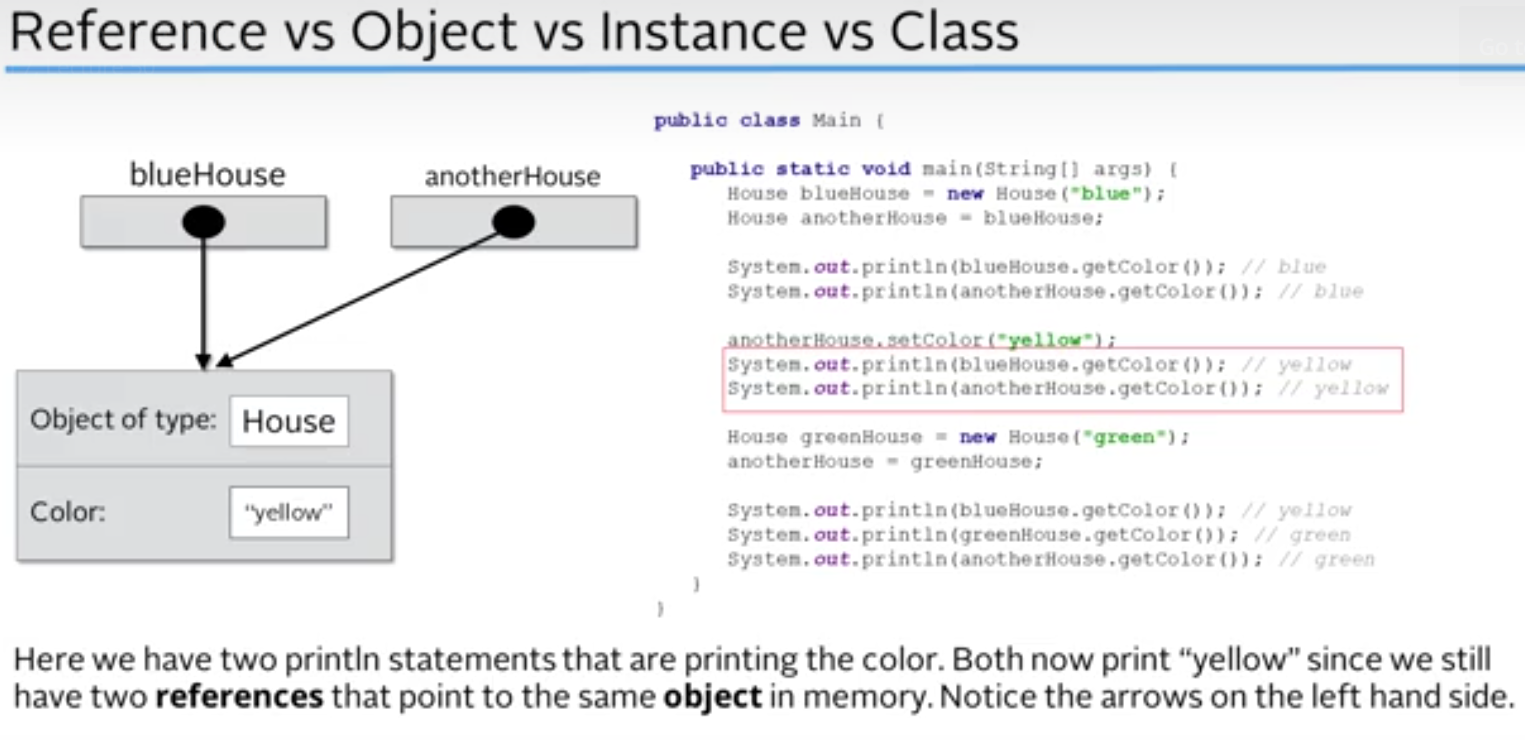


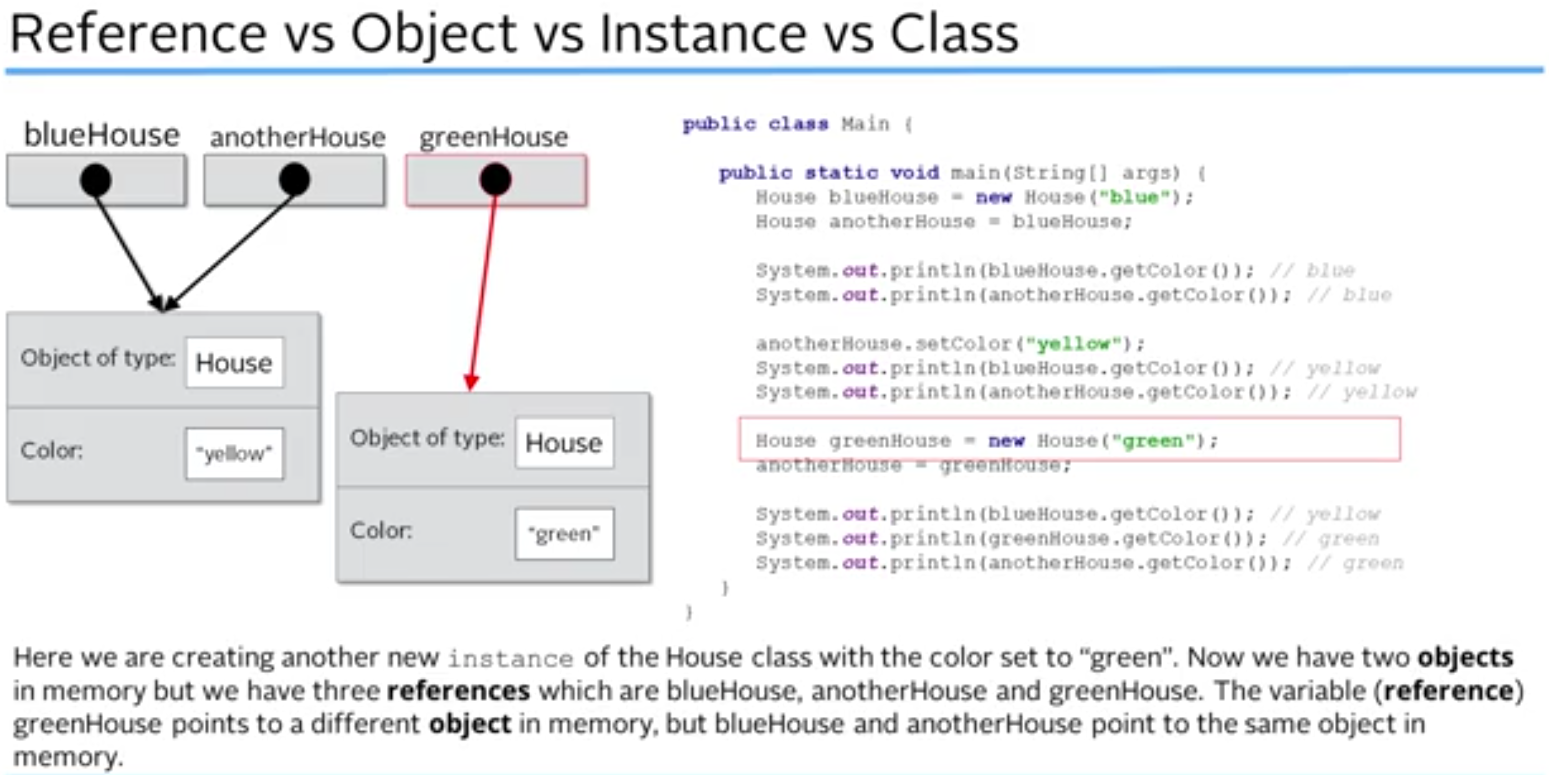


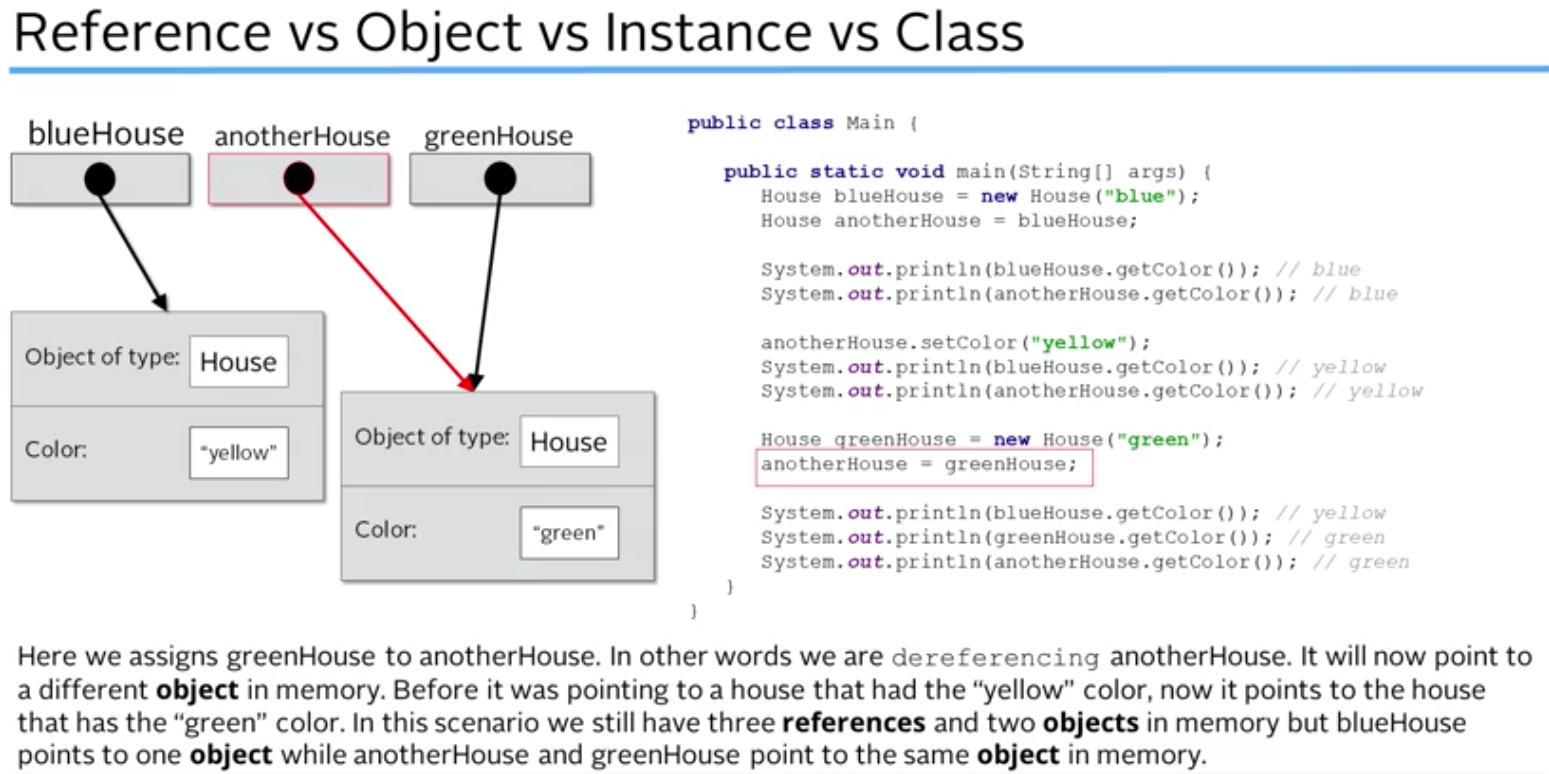
So we have two pieces of paper with the address written on them.

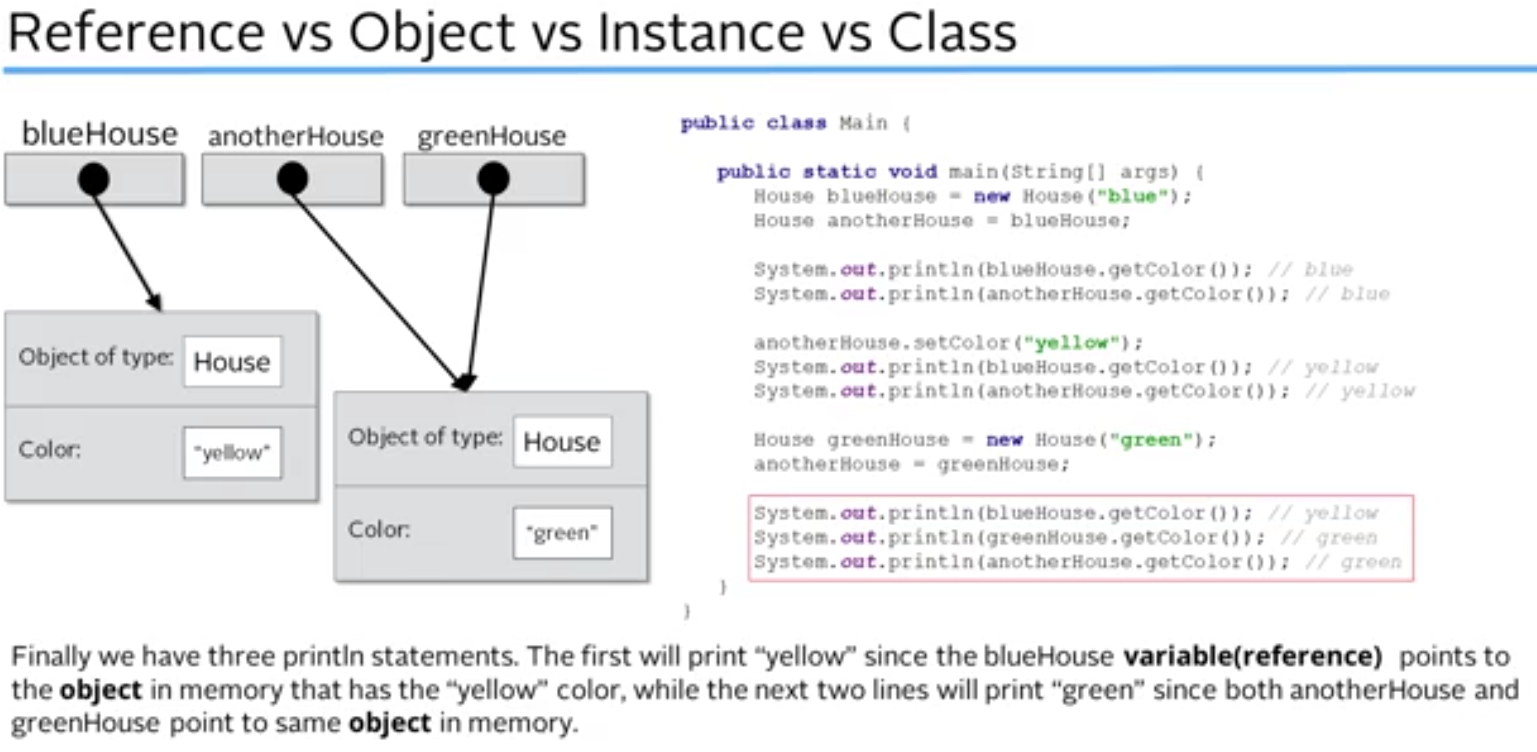




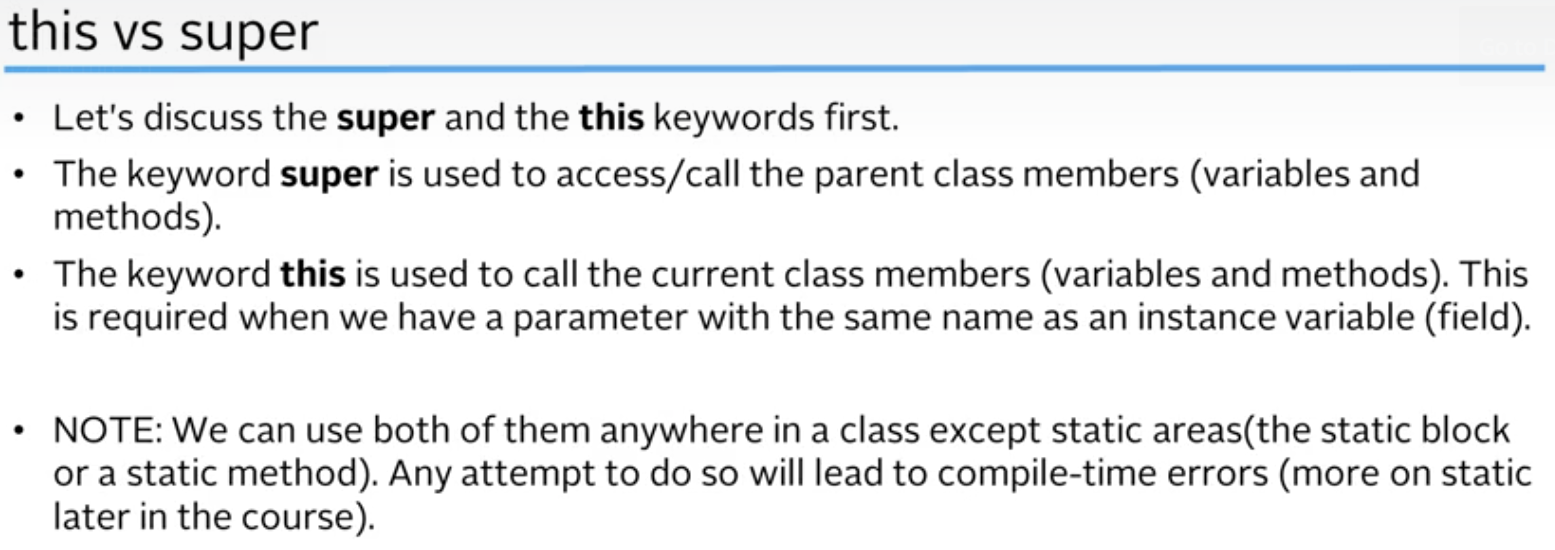


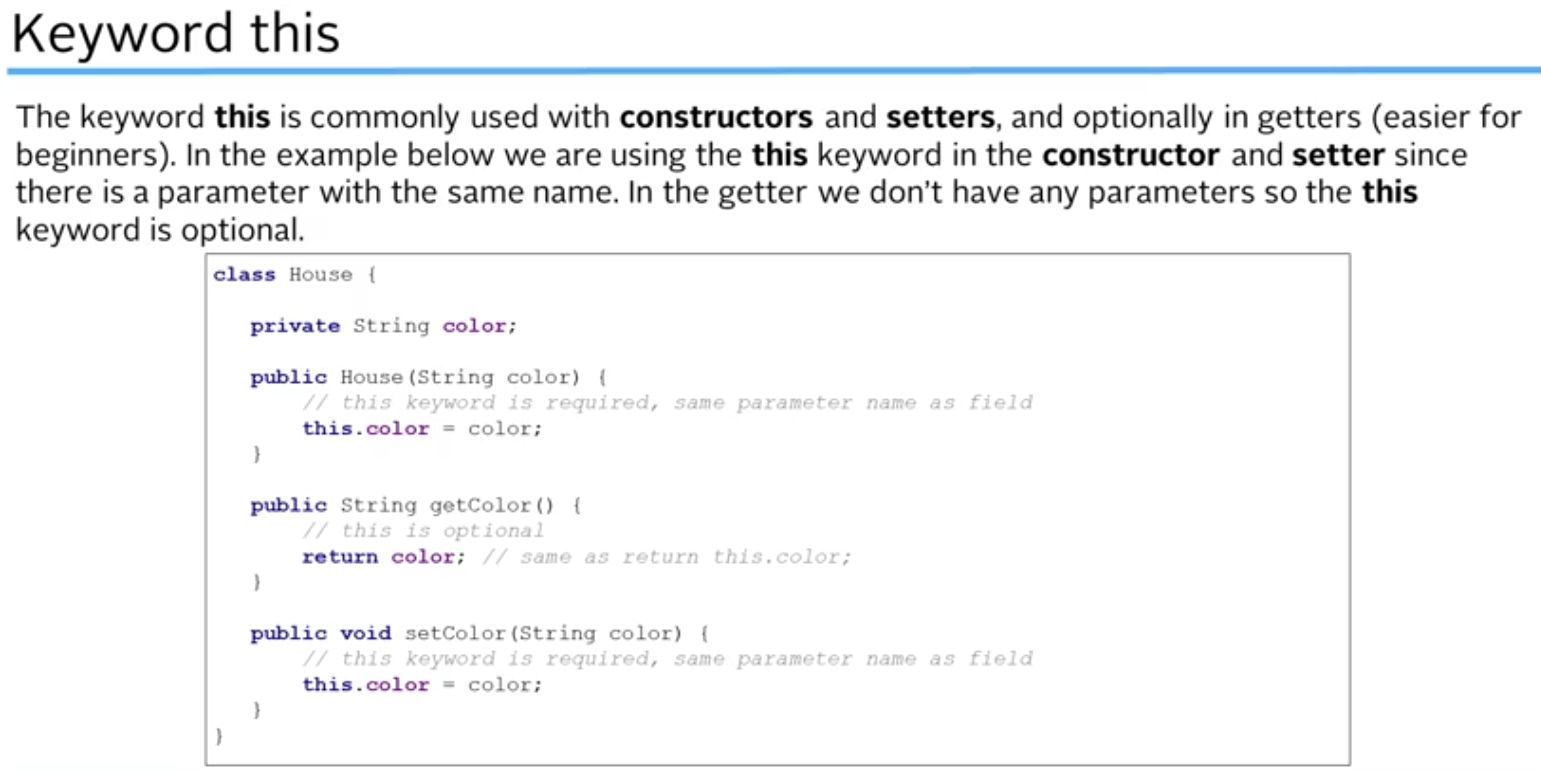


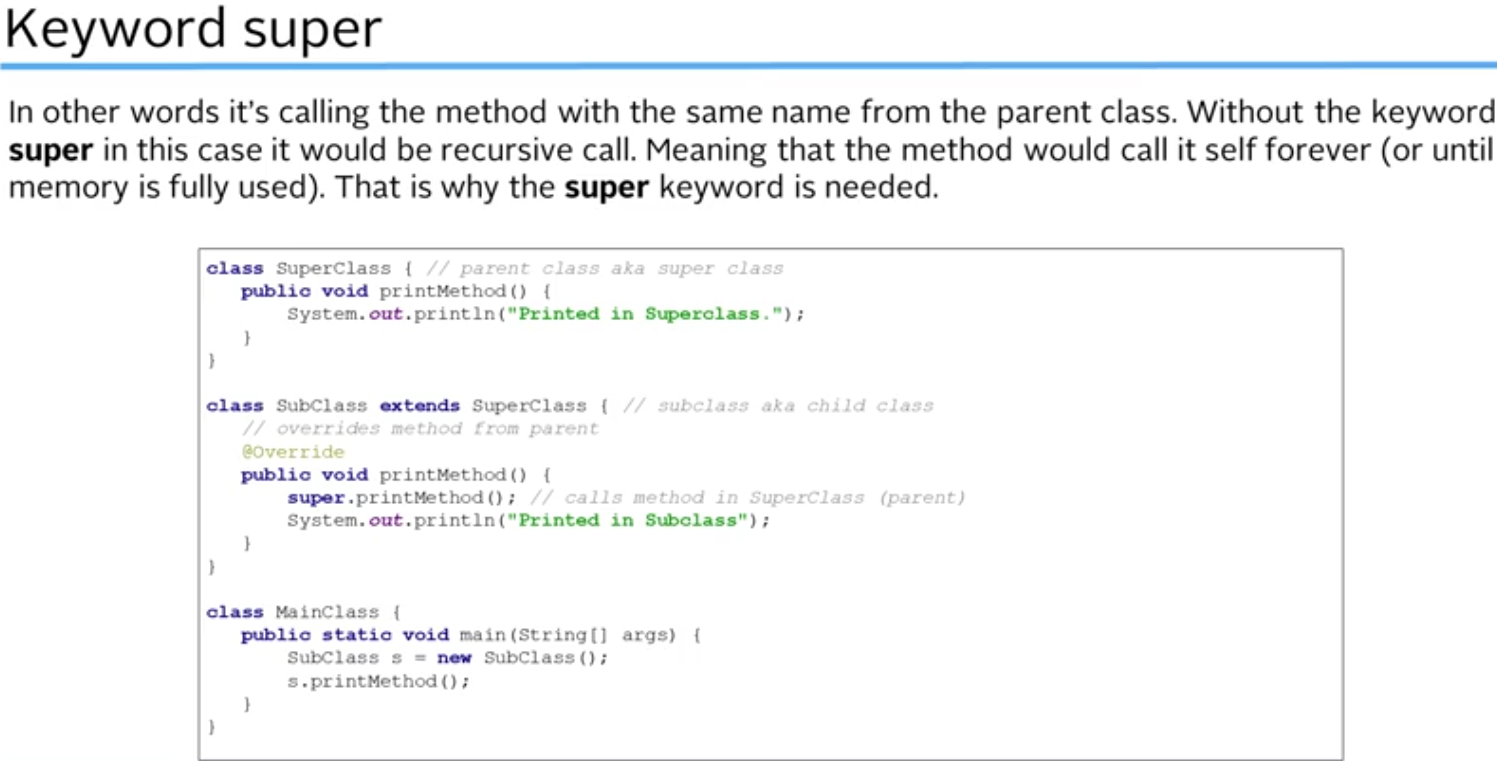


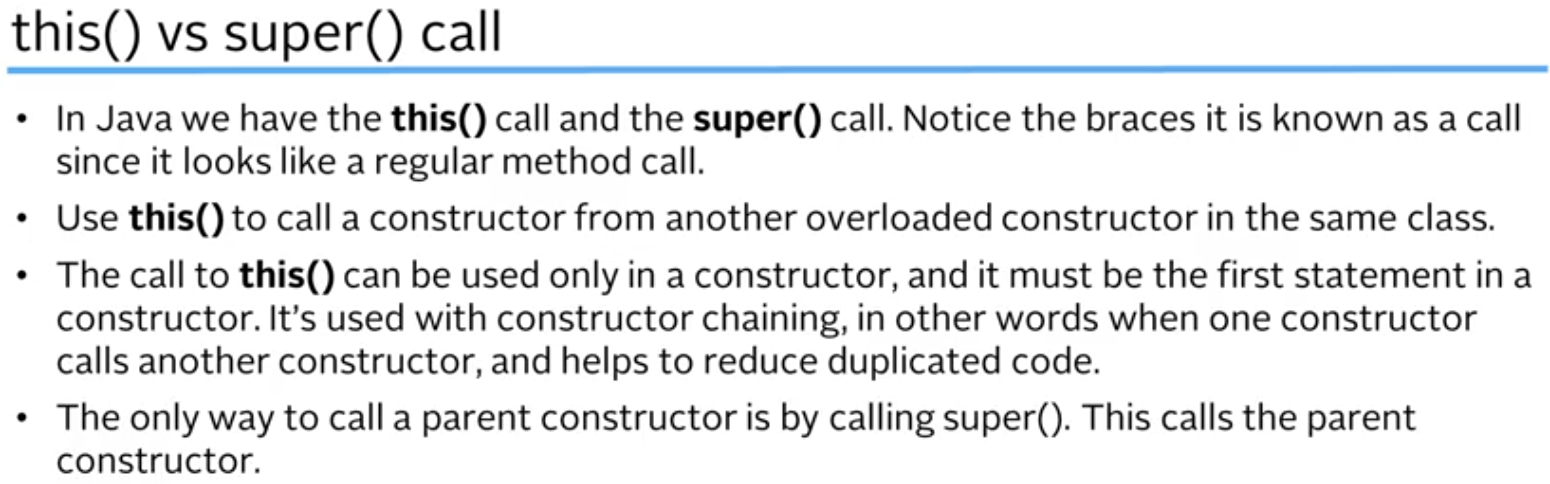


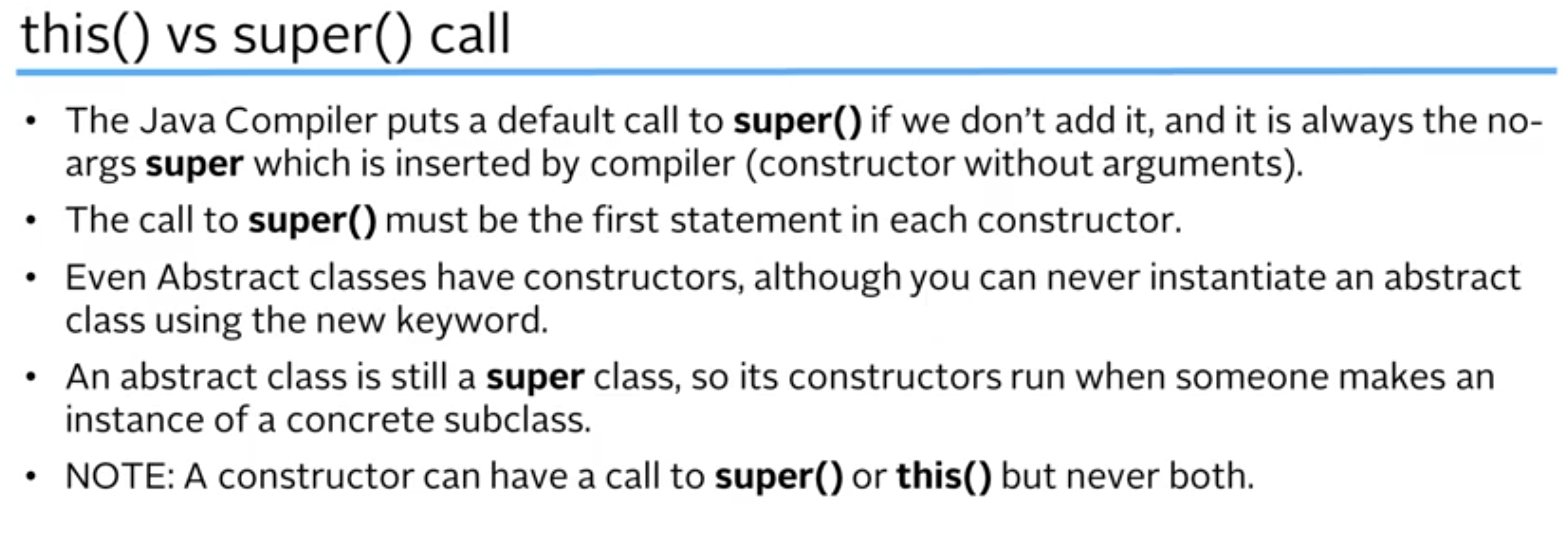
In Java there are always references to deal with. There is no way to access the objects directly. Everything is done using that reference.

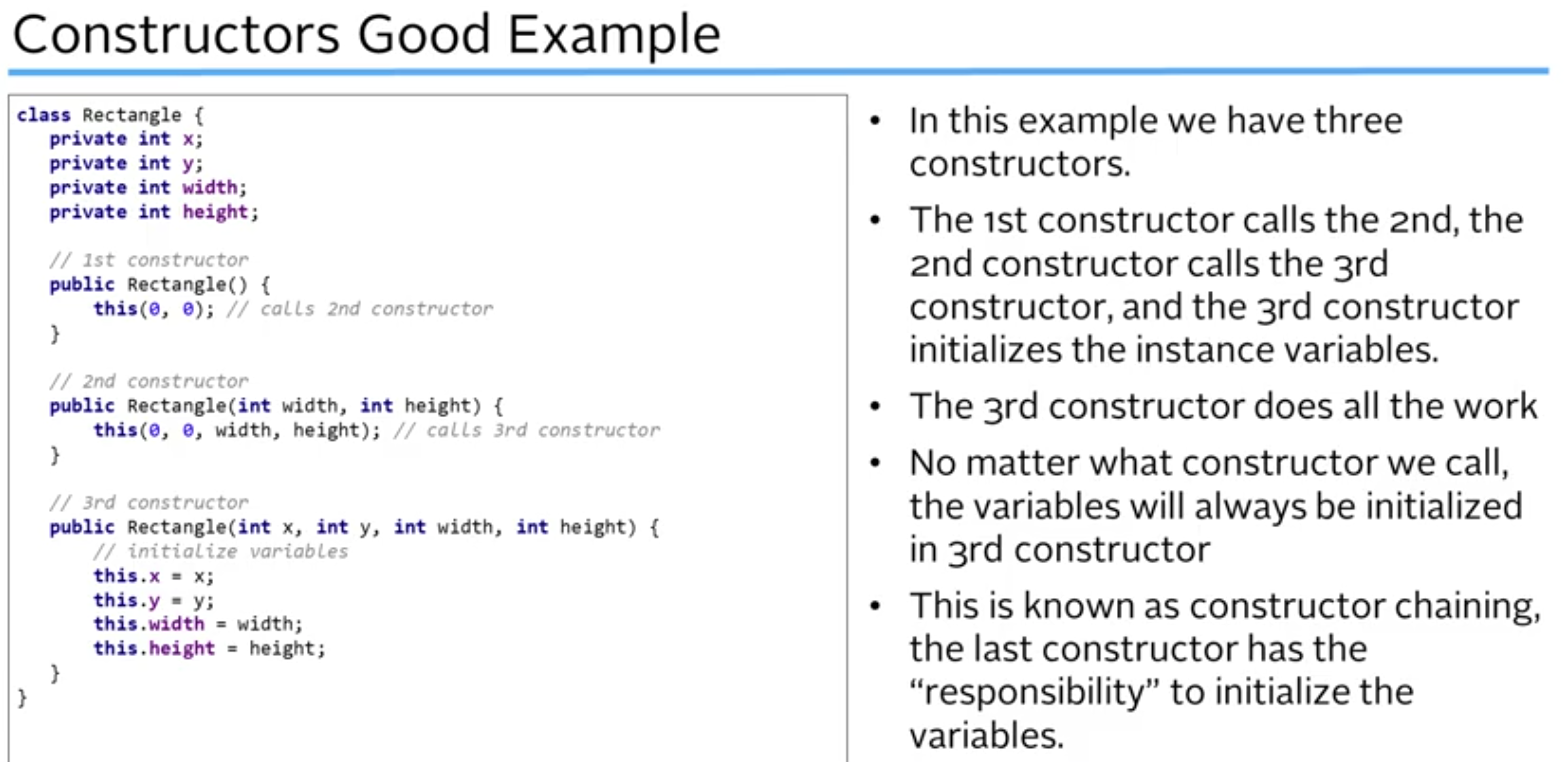


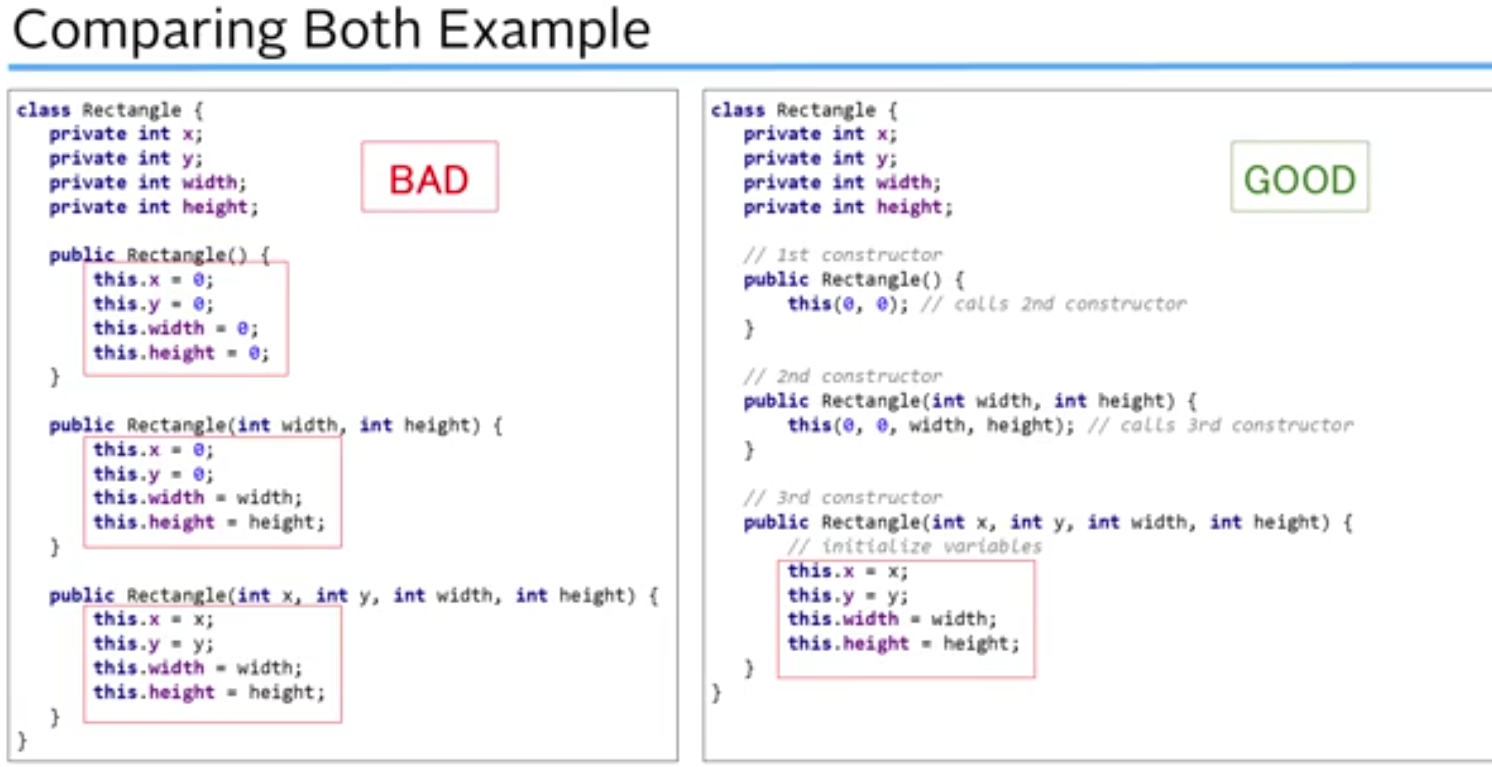




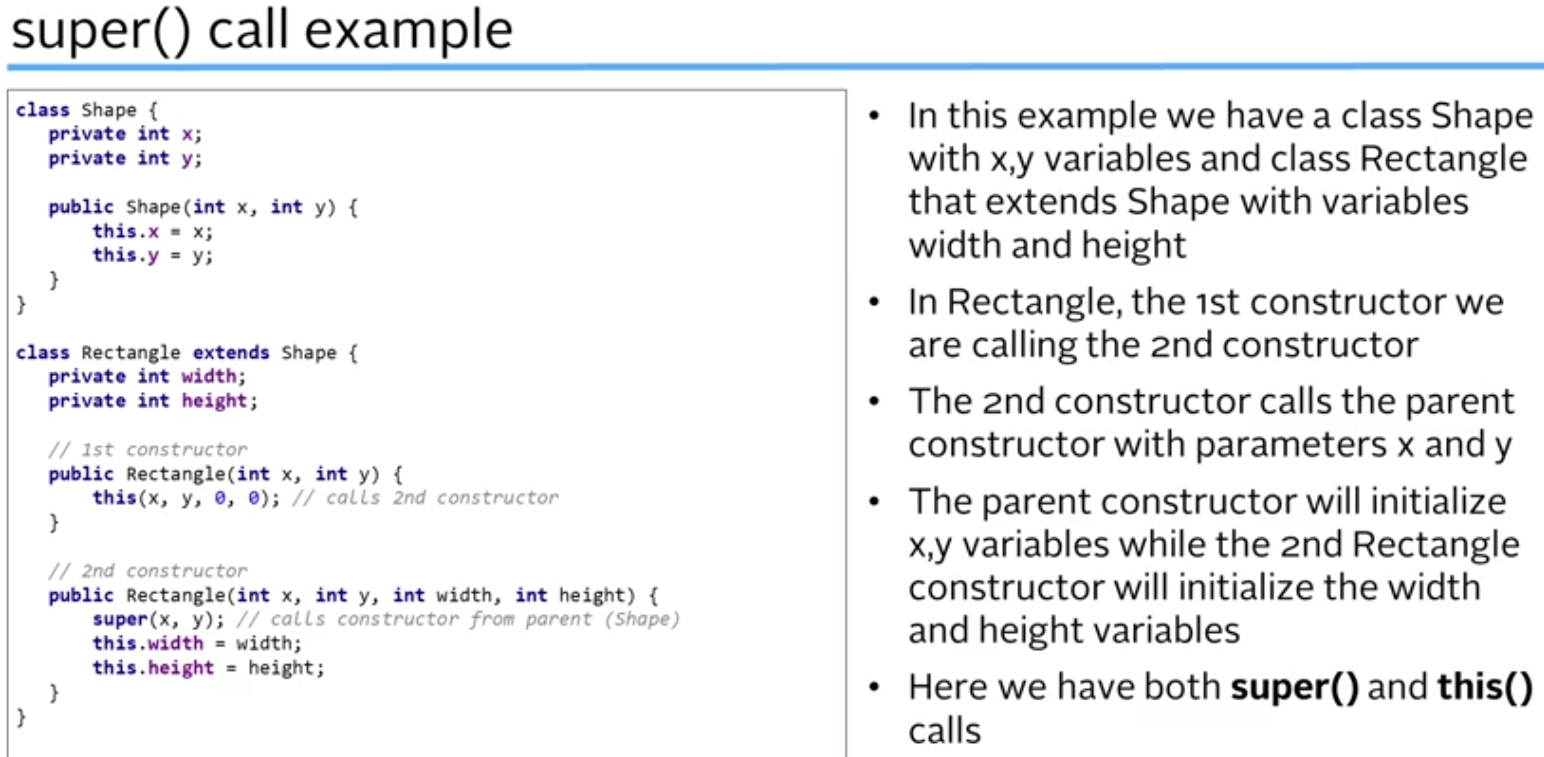








Has lot of duplicated code which leads to more bugs and maintenance



**Inheritance** = IS-A relationship. Dog IS-A Animal.

**Composition** – HAS-A relationship. It modelling parts of a greater whole. Composing a class based on other class (as component).

So Computer HAS-A relationship with Keyboard, Mouse, Monitor, Motherboard, etc.

As a general rule, if there is a situation where Composition and Inheritance both fit in, then prefer Composition over Inheritance because it provides lot of flexibility.

**Polymorphism** – Allows actions to act differently based on the object on which it is called on.

Runtime overriding = Runtime polymorphism: When the method is executed based on the type of object created. If object does not have an implementation of an inherited method() then base class method() implementation is called.

**Arrays** – Is a Data Structure which allows you store values of the same type like array of int, strings, chars, etc. Works for all primitive types and Strings.

dataType[] dataTypeArray = new dataType[sizeOfArray];

Example:

**int**[] intArray = **new** **int**[10];

String[] stringArray = **new** String[20];

Default values of numeric arrays are set to 0.

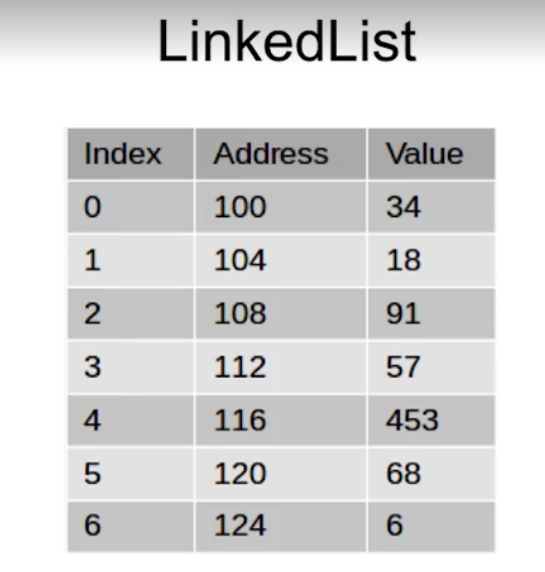
String and other objects are by default initialized to null.

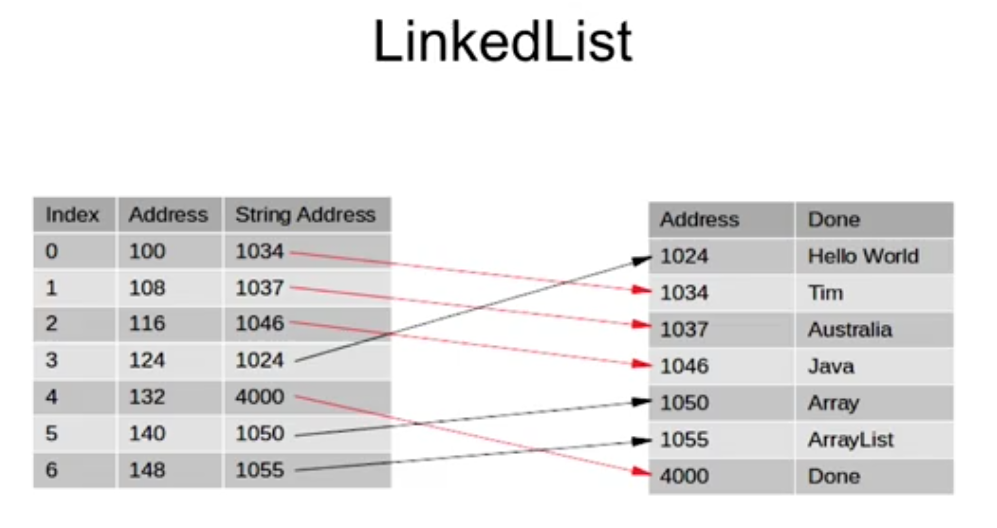
Boolean array is initialized to false.

**Auto Boxing** – Converting a primitive type to Wrapper class type

For int. Java allocates 4 bytes for the size of int

**Unboxing** – Converting the Wrapper class type to primitive type

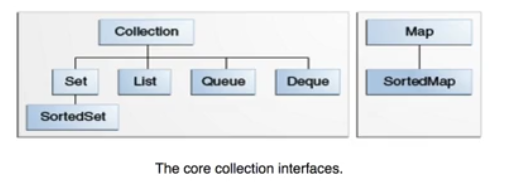




Need not be contiguous.

For Strings: It allocates 8 bytes which refers to the address where the String lives

If we print the list it will print according to the Index -> Address -> String Address



TreeSet – implements SortedSet. It is navigable set. Elements in a treeset must be comparable.

Shallow copy – is when just the reference it copied

Deep copy – is when individual elements are also copied. So this actually creates a different copy altogether.