CP213 Lesson 7: Class Inheritance

* Inheritance is the process by which a new class is created from another class.
* New class is called a *derived class*, original is called a *base class*
* Derived class automatically has all the instance variables and methods from the base class and it can gave additional methods and instance variables as well
* Inheritance is good because it allows code to be reused
* Other than the Object class, every class has a superclass (java supports only one), if none is defined then the default superclass is the object class
* Derived classes can override the methods from the base class as needed
* The return type of a class method can be a descendant class – known as covariant return type
* In a derived class you can make a previously private method public, but not the other way around
* You cannot override “final” methods
* When using the super() keyword in a constructor it must be the first line and you cannot pass instance variables to it
* The object of a derived class has the type of all its ancestor classes
* A derived class object can be passed as a parameter to an ancestor class – polymorphism
* Private instance variables in the base class are not accessible directly by derived classes however they can get copies of them through getters and setters
* A variable stored as “protected” is quite easily accessed – easier to use private
* You can call overridden methods from a super class using super.toString() for example, but this only works for overridden methods
* Super() calls the direct parent method, super.super.toString() is not allowed
* Every object inherits the getClass() method from the object class. This is useful when comparing two objects
* instanceOf() will return true for comparing two objects in the same hierarchy (so a parent can equal a child)

Ex.

Employee e = new Employee("Joe", new Date());

HourlyEmployee h = new HourlyEmployee("Joe", new Date(), 8.5, 40);

boolean testH = e.equals(h);

boolean testE = h.equals(e);

* testH will be **true** because **h** is an **Employee** with the same name and hire date as **e.**However, testE will be **false**, because **e** is not an **HourlyEmployee** and cannot be compared to **h.**
* this would not happen if the getClass() method were used instead
* getClass() can use == and !=

Keywords:

* Ancestor class
* Base class
  + Also called the “super class” the base class has general instance variables and methods that can be extended by its children.
* Child class
  + A class that extends the base class. It inherits all of the methods and instance variables from the base class as well as has its own specific methods and instance variables.
* Composition class
* Covariant
* Derived class
* Descendent class
* Extends
* getClass
  + A method used that returns the class of any given object. This method is able to differentiate between a parent and child class and will not make them equal unlike instanceOf. You can use == and != when comparing two objects returned from this method.
* Has a
* Instance of
  + Returns a Boolean on whether an object is of a given class. Will return true if the object is any part of the class hierarchy and is thus not very useful when overriding the equals() method.
* Is a
* Override
  + Every object inherits methods from the Object superclass such as equals() and toString(). These methods by default aren’t very useful to the user so when you are defining your own classes it is important to rewrite these using the @override to make it more specific to your class.
* Parent class
* Sub-class
* Super-class