CP213 Lesson 8

Encapsulation: a mechanism of wrapping the instance variables and methods or actions together as a single unit called a class.

Inheritance: allows a base class to be defined, and other classes to be derived from it. The code for the base class can then be used for its objects, as well as objects of any derived classes.

Polymorphism: allows changes to be made to method definitions in the derived classes, and have those changes applied to the software written for the base class. In other words, Polymorphism is the ability to associate many meanings to one method name. It does this through a special mechanism knows as *late binding* or *dynamic binding.*

* If a method definition is associated with its invocation when the code is compiled that is called early binding
* If the method definition is associates with its invocation when the method is invoked (run time) that is called late binding or dynamic binding.
* Java uses late binding for all methods (except private, final, and static methods)
* Static methods do not have late binding
* A method marked final indicates that it cannot be overridden
* A class marked final indicates that it cannot be used as a base class from which to derive any other classes
* Upcasting is when an object of a derived class is assigned to a variable of a base class (or any ancestor class)
* Downcasting is the opposite, you have to be careful as this will often produce errors
* The clone() method is inherited from the object class and is supposed to return a deep copy of the calling object, it is necessary to override this method.

Ex.

public Employee clone() {

return new Employee(this);

}

* Before Java 5.0 the clone method would always return the “Object” type, so it was necessary to typecast it. Although no longer needed it’s still good practice when in doubt

Employee copy = (Employee)original.clone();

* Although the clone and copy methods appear to do the same thing, sometimes the clone method will be more effective at creating a true copy when there are instances of child classes involved, since the clone method uses late binding that can be executed at runtime.
* copyArray[i] = new Employee (emp[i] ) ;

creates a copy of a plain Employee as opposed to a manager

copyArray[i] = emp[i].clone() ; // Manager object is copied

* the reason this works is because the class and each subclass has the clone method, so polymorphism is used.
* Abstract method: acts as placeholder in the base class for a method to be defined in a future subclass

Ex.

public abstract double getPay();

* A class that has at least one abstract method is called an abstract class and must have the modifier abstract included in its class heading
* A non abstract class is called a concrete class.
* You cannot create an instance of an abstract class – every employee is a manager, team leader, etc. there is no point of just having an employee object
* Abstract class
  + A class containing at least one abstract method, instances of this class cannot be created.
* Abstract method
  + A method in a base class that acts as a placeholder for future child classes to implement.
* Clone method
  + All classes inherit this method from the Object class. When properly overridden this acts as a better way to copy objects since it can use polymorphism and late binding to copy objects at runtime resulting in true copies.
* Concrete class
  + The opposite of an abstract class, a concrete class is simply a class with no abstract methods.
* Downcasting
* Early binding  or static binding
* Late binding or dynamic binding
* Polymorphism
* Upcasting
  + Upcasting is when an object of a derived class is assigned to a variable of a base class (or any ancestor class)