# CIS 3145 Class Notes: Text Chapter 11

## Inheritance Concepts

**Objectives**

* Describe inheritance
* Describe the use of inheritance in Java
* Know how to create super classes and sub classes
* Know how to compare and cast objects
* Know how to use the Abstract and Final keywords

**Inheritance**

**Inheritance** allows us to easily create new classes based on the **fields**, **constructors**, and **methods** of existing classes.

The original class is called the **base**, **parent**, or **super** class.

The new class is called the **derived**, **child**, or **sub** class.

This new class **extends** the original class because we can add on new fields, constructors and methods in addition to **overriding** the elements of the original class.

All classes inherit from the **Object** class in the java.lang namespace. This is the “great, great, great…grandparent” of all classes. <https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html>

**Object Class**

The toString() and equals() methods are overridden to customize the behavior of the child class.

The getClass() method returns the type of class the object is.

The clone() method makes a copy of the object.

The ‘**garbage collector’** automatically collects and removes objects from memory that are not being used. Before the object is deleted the **finalize**() method is called.

**Generic Superclasses**

The ‘Product’ class contains elements (fields, constructors, and methods) that apply to ALL products. The subclasses do not have to define these elements themselves.

**Creating and Using a Superclass**

Access Modifiers**:** There are 4 modifiers for object elements. This relates to **encapsulation** because it regulates how data gets into and out of the object.

public 🡪 Available to everyone

protected 🡪 Neighborhood: available in the same subclass and package

‘No modifier’ 🡪 Street Block: available in the same package

private 🡪 Only in the current class

To override a method, use the “@**Override**” annotation.

**Creating and Using a Subclass**

The first element is to declare a new class with the “**extends**” keyword.

Then we will want to run a constructor for the parent class as well as the constructor elements of our subclass. To call the parent class use the “**super** ()” constructor. The “**super**” keyword also works as a modifier for the methods and variables of the parent class.

**Polymorphism with inheritance**

A variable for the parent class can be used to hold objects from the child class. Thus methods can use a generic parent class as the Parameter data type and any child can be passed in.

public static void myMethod (Product p) {

//code here

}

Product p = new Product();

Book b = new Book();

call myMethod (p); 🡨 The parameter data type is Product for myMethod,

call myMethod (b); 🡨 but a Book object is okay because it is a child of Product

*We can assign a child class to a parent variable*:

Product p = new Product();

Book b = new Book();

p = b;

*We can define a product variable as a child data type:*

Product p =new Book();

**Object Types**

All objects are managed by the java “**Class**” class. Each object has an associated “**Class**” object.

* Found in the namespace: java.lang.Class
* Runtime type identification (RTTI)
* **getName**() is a common method for this class

The method **getClass**() returns the data type of the object AS AN OBJECT!!! The ‘data type’ is in essence the name of the class.

Casting Objects

**Implicit** casting for a child assigned to a parent. The child will have the parent details.

p = b;

**Explicit** casting is **REQUIRED** for a parent assigned to a child. The child *might* have details that are missing in the parent.

b = (Book) p;

Comparing Objects

Use: product\_A.equals (Product\_B) to test if the objects share the same ***Memory Location***

**Create** an ‘equals’ method for your class to test if the ***Data*** is the same

* Test if ALL **instance** **variables** in the two objects are the same

Abstract Keyword

* Abstract Class: Like a **template**, cannot create an object, can ONLY use inheritance
* Abstract Method: Create the method **declaration** without any **code.** This is a promise to include certain methods.

Final Keyword

* public **final** *class* XYZ -> Class can NOT be inherited
* pubic **final** double *MethodX* -> the Method can NOT be overridden
* **final** int *myIntVariable* -> creates a Constant