**Part I**

**Kingdom.java**

**package** finalassignment;

**public** **class** Kingdom { // Kingdom

Kingdom x = **new** Kingdom();

**public** **void** setName(String newName) {

kingdomName = newName;

}

**public** **void** King(**int** numberOf) {

itemName = numberOf;

}

**public** **void** printItem() {

System.***out***.println(itemName + " " + itemName);

}

**public** Kingdom() { // Constructor

kingdomName = "Jon's Castle";

itemName = 1;

}

**public** String getKingdomName() {

**return** kingdomName;

}

**public** **void** setKingdomName(String kingdomName) {

**this**.kingdomName = kingdomName;

}

**public** String toString() {

**return** (String.*valueOf*(x));

}

**protected** **int** itemName;

**protected** String kingdomName;

}

**City.java**

**package** finalassignment;

**public** **class** City **extends** Kingdom { //

City x = **new** City();

**public** **void** government(String typeOf) {

classification = typeOf;

}

**public** String getExpiration() {

**return** classification;

}

**public** City() { // Constructor

classification = "Jonsville";

}

**public** String getClassification() {

**return** classification;

}

**public** **void** setClassification(String classification) {

**this**.classification = classification;

}

**public** String toString() {

**return** (String.*valueOf*(x));

}

**private** String classification;

}

**Downtown.java**

**package** finalassignment;

**public** **class** Downtown **extends** Kingdom {

Downtown x = **new** Downtown();

**public** **void** government(String typeOf) {

classification = typeOf;

}

**public** String getExpiration() {

**return** classification;

}

**public** Downtown() { // Constructor

classification = "Partle Street";

}

**public** String getClassification() {

**return** classification;

}

**public** **void** setClassification(String classification) {

**this**.classification = classification;

}

**public** String toString() {

**return** (String.*valueOf*(x));

}

**private** String classification;

}

**Bank.java**

**package** finalassignment;

**public** **class** Bank **extends** Downtown {

Bank x = **new** Bank();

**public** **void** DaysofOpen(String availability) {

schedule = availability;

}

**public** String whenItOpens() {

**return** schedule;

}

**public** Bank() { // Constructor

schedule = "Weekends M-F 9am-5pm";

}

**public** String getSchedule() {

**return** schedule;

}

**public** **void** setSchedule(String schedule) {

**this**.schedule = schedule;

}

**public** String toString() {

**return** (String.*valueOf*(x));

}

**private** String schedule;

}

**Hospital.java**

**package** finalassignment;

**public** **class** Hospital **extends** City {

Hospital x = **new** Hospital();

**public** **void** Capacity(String people) {

classification = people;

}

**public** String typeOf() {

**return** classification;

}

**public** Hospital() { // Constructor

classification = "Service Hospital";

}

**public** String getClassification() {

**return** classification;

}

**public** **void** setClassification(String classification) {

**this**.classification = classification;

}

**public** String toString() {

**return** (String.*valueOf*(x));

}

**private** String classification;

}

**Towncenter.java**

**package** finalassignment;

**public** **class** Towncenter **extends** Downtown {

Towncenter x = **new** Towncenter();

**public** **void** DaysofOpen(String availability) {

schedule = availability;

}

**public** String whenItOpens() {

**return** schedule;

}

**public** Towncenter() {// Constructor

schedule = "Weekends M-F 9am-5pm";

}

**public** String getSchedule() {

**return** schedule;

}

**public** **void** setSchedule(String schedule) {

**this**.schedule = schedule;

}

**public** String toString() {

**return** (String.*valueOf*(x));

}

**private** String schedule;

}

**Supermarket.java**

**package** finalassignment;

**public** **class** Supermarket **extends** City {

Supermarket x = **new** Supermarket();

**public** **void** Capacity(String people) {

classification = people;

}

**public** String sizeOf() {

**return** classification;

}

**public** Supermarket() { // Constructor

classification = "Big Supermarket - 105,000 sq ft";

}

**public** String getClassification() {

**return** classification;

}

**public** **void** setClassification(String classification) {

**this**.classification = classification;

}

**public** String toString() {

**return** (String.*valueOf*(x));

}

**private** String classification;

}

**Employees.java**

**package** finalassignment;

**public** **abstract** **class** Employees **extends** Kingdom **implements** Sims {

**protected** **int** x;

**public** **abstract** **void** print();

**public** **void** setX(**int** a) {

x = a;

}

**public** **int** getX() {

**return** x;

}

**public** Employees() { // Constructor

x = 0;

}

**public** String toString() {

**return** (String.*valueOf*(x));

}

}

**Sims.java**

**package** finalassignment;

**public** **interface** Sims {

**public** **void** windowOpened(); // Program opens, if library is available

}

**Part II**

Describe the differences between abstract classes and interfaces. Give an example of where you might use an interface in a program. Give an example of where you might use an abstract class in a program.

When you might use interface in a program, for example, is for opening, closing, activating, or deactivating a window. It’s a type of class that contains only abstract methods or named constants. It uses the mechanism of *implements.*

As for an abstract class, you can instantiate an object of a subclass of an abstract class but only if the subclass gives the defintions of all the abstract methods of the superclass.

**Part III**

In your own words, explain the following object-oriented programming principles:

 abstraction

 encapsulation

 inheritance

 polymorphism

Polymorphism allows us to use a subclass within another class. As if we to create a hierarchy of what determines a superclass and what governs it from underneath.

Inheritance adopts traits and functions from the other class in belongs to. Of a class to the existing class. Without making any physical changes.

Abstraction is from providing code without a body*.* The same goes for an abstract method, it only has the heading and no body.

Encapsulation has to do with identifying the object and identifying the data that goes with it.