**Assessment Task**

Exercise 1.

Describe the four key Object Orientated Programming principals.

**Encapsulation:**

In Java a way of protecting data is by hiding some of the variables used in a program. This is done by Encapsulation where you would group a set of variables in a private class to have them hidden. From that private class you then group a few methods where you can access the data stored in those variables in some public classes. The methods are usually created by Javas get and set methods.

Please see example below:

**public** **class** Dog {// Main class

**private** String name; //setting up variables

**private** **int** age;

**private** String breed;

**public** String getName() {//get method to get the data from the variable

**return** name;

}

**public** **int** getAge() {

**return** age;

}

**public** String breed() {

**return** breed;

}

}

In the above example you will see that in the main public class we set up the variables by giving them a label with a type of data i.e. String or integer.

**Inheritance:**

Inheritance is where the program can inherit new methods or fields into a class from another class. This enables us to group certain variables together but giving use to be able to aces them all for a method or function. Also enables us to expand the data out even further. This gives us room for improvement later on. This method also allows us to edit an original method and override it with a new method.

Please see example below:

**class** Animal{}//first class is animals

**class** Pet **extends** Animal{ 2nd class is pet but also includes first class.

}

**public** **class** Dog **extends** Pet{ 3rd class is dog and extends Pet

**private** String name;

**private** **int** age;

**private** String breed;

**public** String getName() {

**return** name;

}

**public** **int** getAge() {

**return** age;

}

**public** String breed() {

**return** breed;

}

**public** **static** **void** main(String args[])

{

Animal a = **new** Animal();//creates new instances of classes.

Pet m = **new** Pet();

Dog d = **new** Dog();

System.***out***.println(m **instanceof** Animal);//Boolean statement to show that pet has inherited class animal.

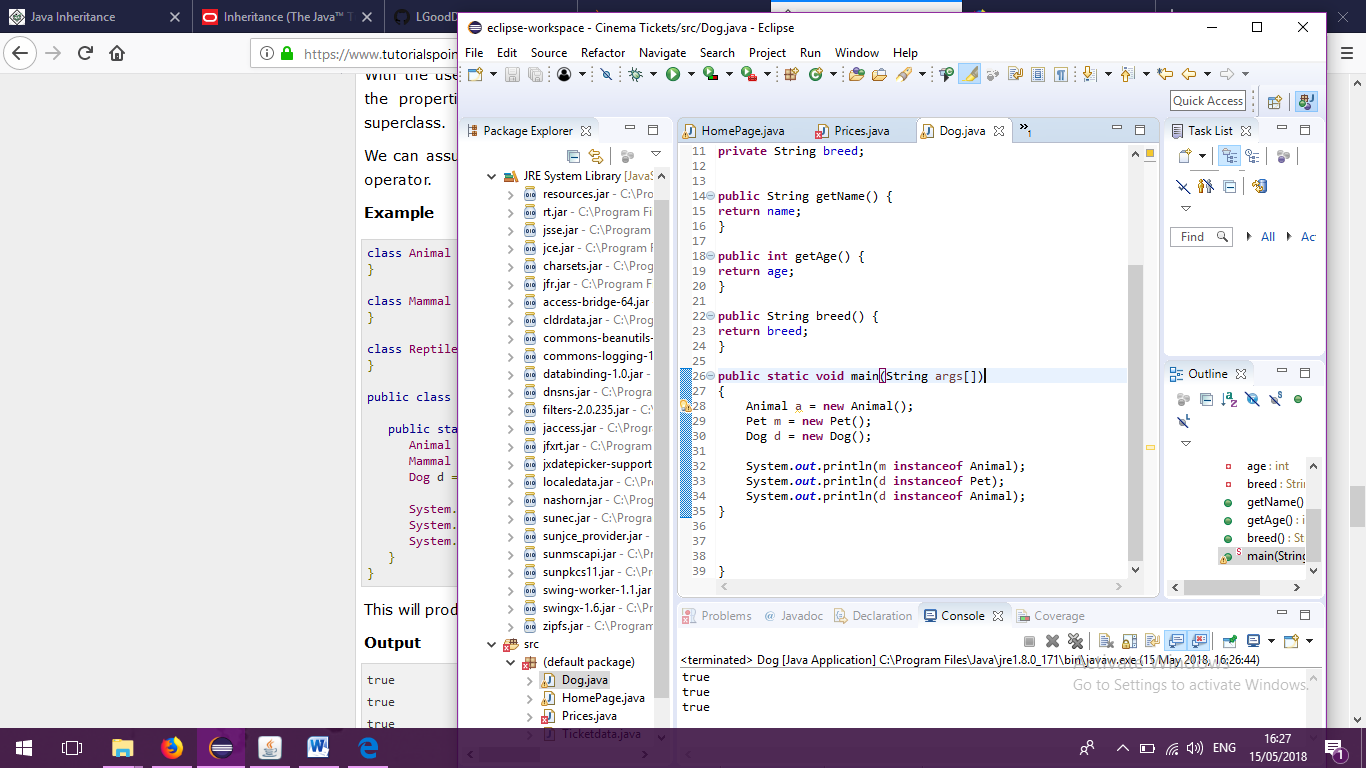
System.***out***.println(d **instanceof** Pet);

System.***out***.println(d **instanceof** Animal);

}

}

Results on console:



From the above example you can see that the first class is inherited by the nd class and the second by the third class.

**Polymorphism:**

This is where an Object in Java has the ability to have many different forms of relationships. This allows a reference in the parent class to call upon an object in a child class. The only way the parent reference can get the object in the child is by calling a reference variable. Any object that can have multiple inheritances or is-a relationship is classed as polymorphic.

Please see example: (Dog.java)

**class** Animal{}

**class** Pet **extends** Animal{

}//unrelevant in this example.

**public** **class** Dog **extends** Pet{

**private** String name; //seting up variable names

**private** **int** age;

**private** String breed;

**public** Dog(String name, **int** age, String breed) //setting up constructors for / /variables with reference to use /the methods.

{System.***out***.println("Getting dog details..");

**this**.name = name;

**this**.age = age;

**this**.breed = breed;

}

**public** **void** DogData() {

System.***out***.println("Dog details are " + **this**.name + **this**.age+ **this**.breed );

} //Message for the console with get methods to get data added in main static.

**public** String getName() {

**return** name;

}

**public** **int** getAge() {

**return** age;

}

**public** String getBreed() { //methods to call data.

**return** breed;

}

**public** **static** **void** main(String args[])

{

}

}

Other File: Medication.java

**public** **class** Medication **extends** Dog {//calss extends original java file (dog).

**private** String medicine;

**public** Medication(String name, **int** age, String breed , String medicine) {

**super**(name, age, breed);

**this**.medicine = medicine; // New constructor similar to other except for added a new value of medicine. Using super to inherit original constructors.

}

**public** **void** DogData() {

System.***out***.println(getName() + getAge() + getBreed() + getMeds());

}

**public** String getMeds(){

**return** medicine;

}

**public** **static** **void** main(String[]args) {

Dog a = **new** Medication("Rover", 5, "GoldenRetriver","none");

Medication b = **new** Medication("Bobby",7,"Terrier","Fluxocillian");

System.***out***.println("New Dog details are...");

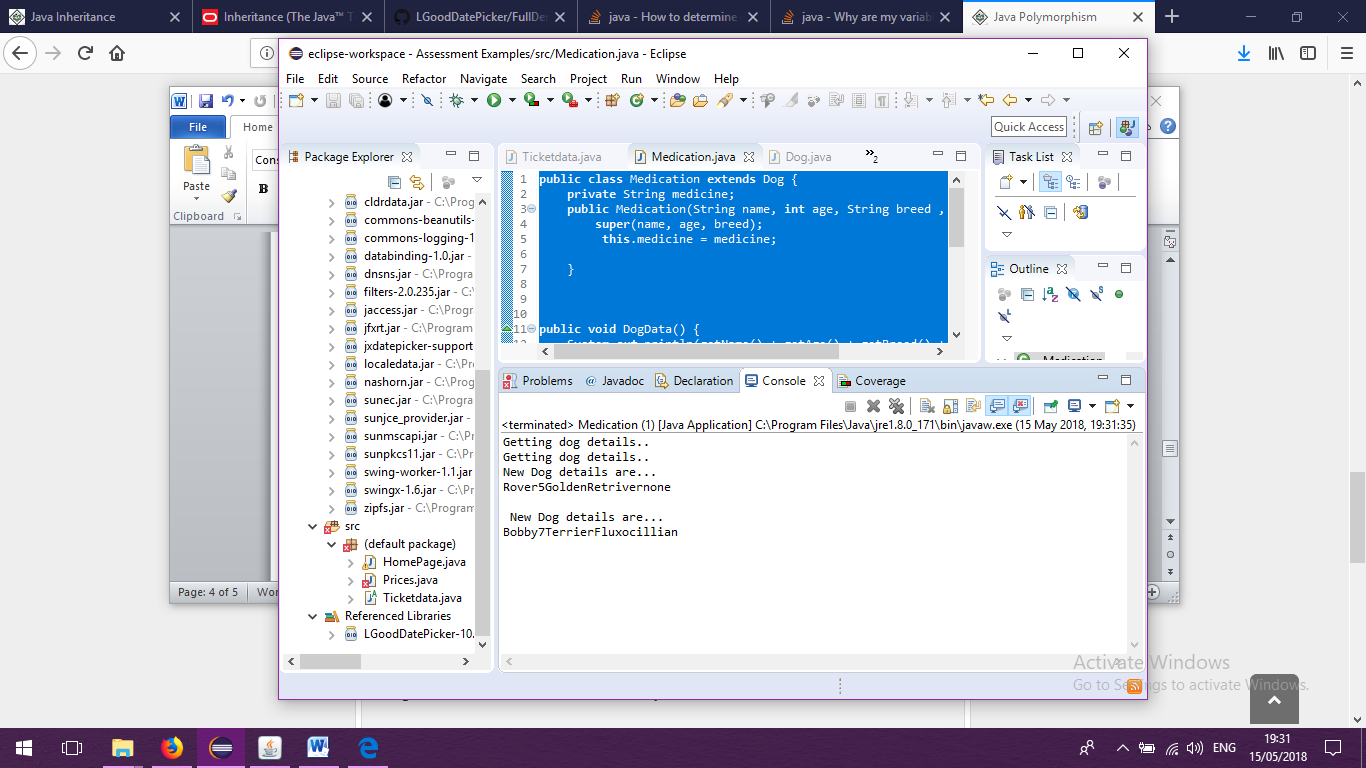
a.DogData();

System.***out***.println("\n New Dog details are...");

b.DogData();

}

}



As you will see from this example of polymorphism that the second file inherits the methods and variables form the original file of age, name and breed. After to which we have added a new variable with a new call method of medicine. In the console you will see that the original print out called DogData is overridden by the new system print out containing the added element of medicine.

**Abstract:**

An abstract class is a class where you can hide the processing side of the program so the only part of the program that the user requires is the main output. For example you just want to know how much something costs not how the program calculated it. This is where you would place the methods that help you get or change the data in the program. An abstract class cannot be instantiated and need subclasses to provide implementation methods to get the abstract method.

Please see example below:

**class** Animal{}

**class** Pet **extends** Animal{

}

**public** **abstract** **class** Dog **extends** Pet{//you can see I have named this an abstract class to hide the methods invoked below.

**private** String name;

**private** **int** age;

**private** String breed;

**public** Dog(String name, **int** age, String breed)

{System.***out***.println("Getting dog details..");

**this**.name = name;

**this**.age = age;

**this**.breed = breed;

}

**public** **void** DogData() {

System.***out***.println("Dog details are " + **this**.name + **this**.age+ **this**.breed );

}

**public** String getName() {

**return** name;

}

**public** **int** getAge() {

**return** age;

}

**public** **class** Medication **extends** Dog {

**private** String medicine;

**public** Medication(String name, **int** age, String breed , String medicine) {

**super**(name, age, breed);

**this**.medicine = medicine;

}

**public** **void** DogData() {

System.***out***.println(getName() + getAge() + getBreed() + getMeds());

}

**public** String getMeds(){

**return** medicine;

}

**public** **static** **void** main(String[]args) {

Dog a = **new** Medication("Rover", 5, "GoldenRetriver","none");

Medication b = **new** Medication("Bobby",7,"Terrier","Fluxocillian");

System.***out***.println("New Dog details are...");

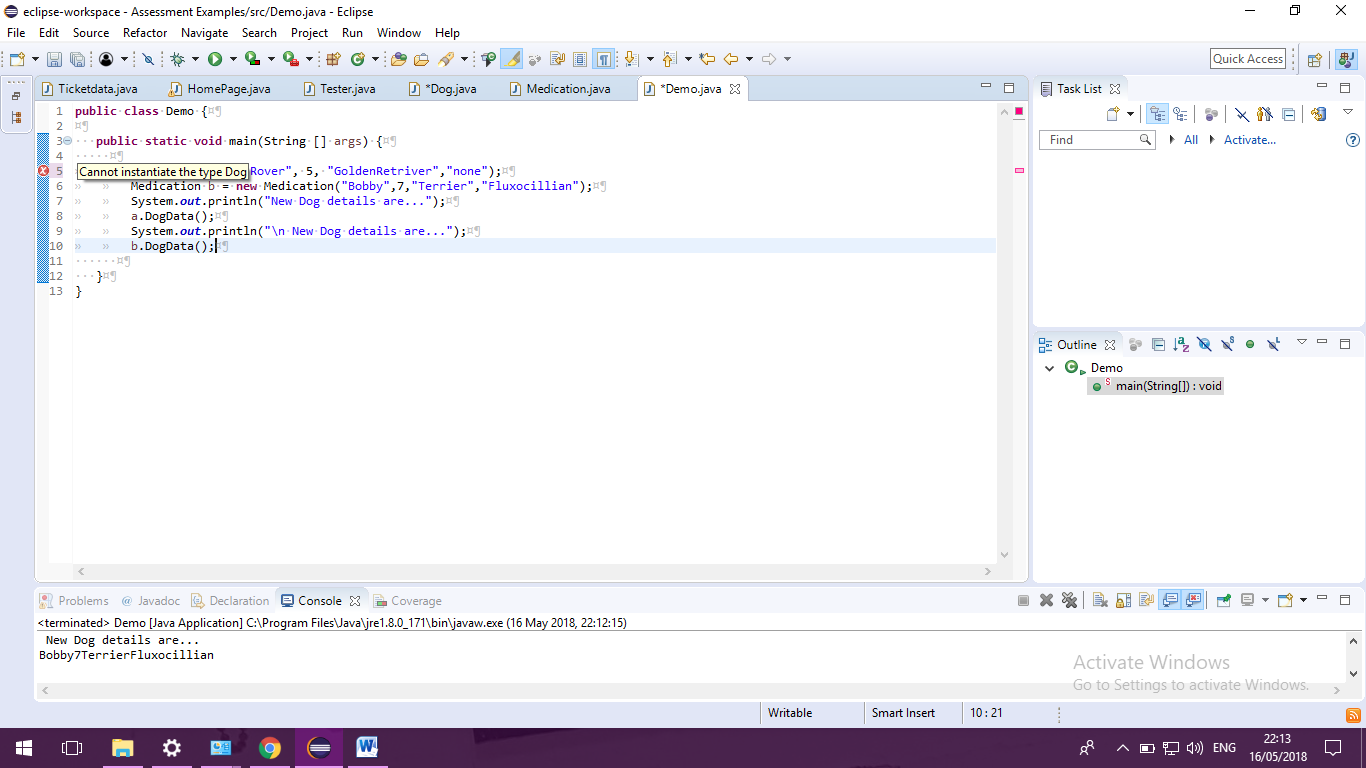
a.DogData();

System.***out***.println("\n New Dog details are...");

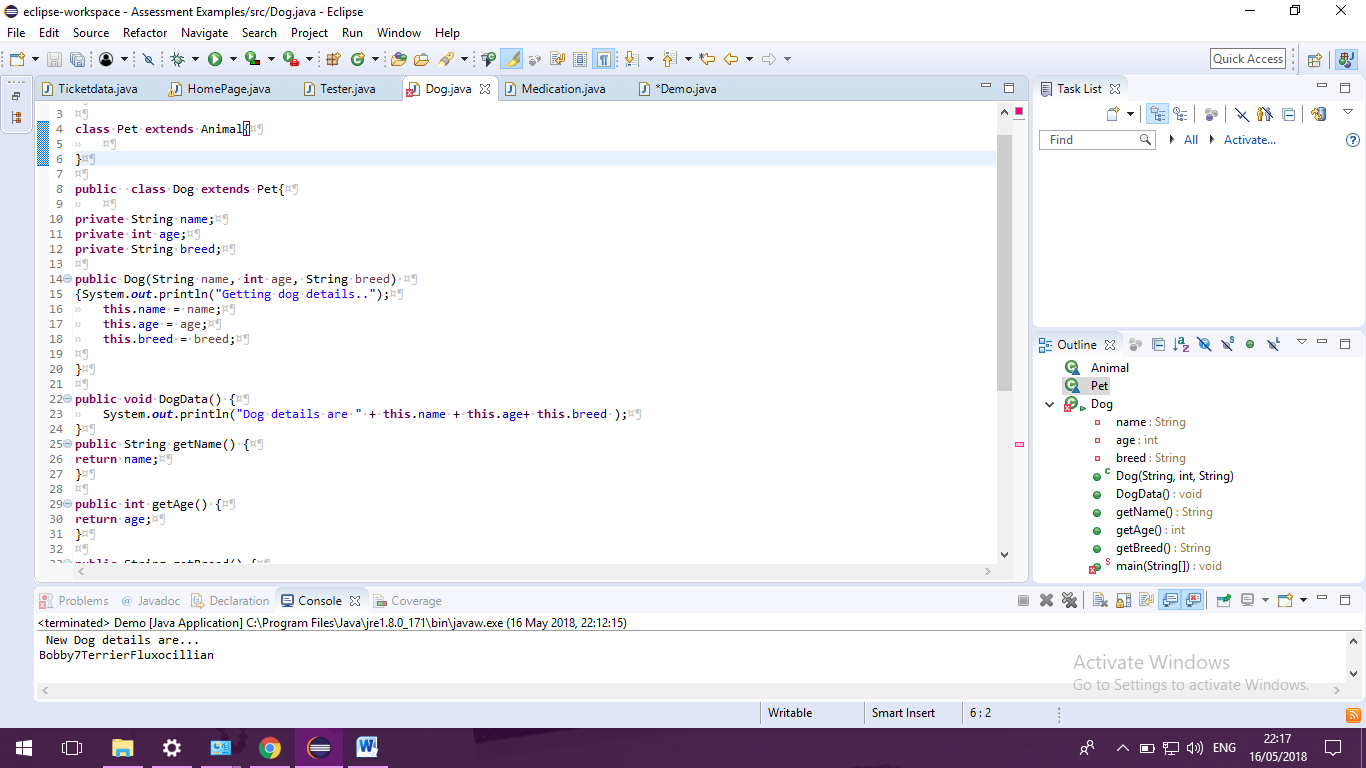
b.DogData();

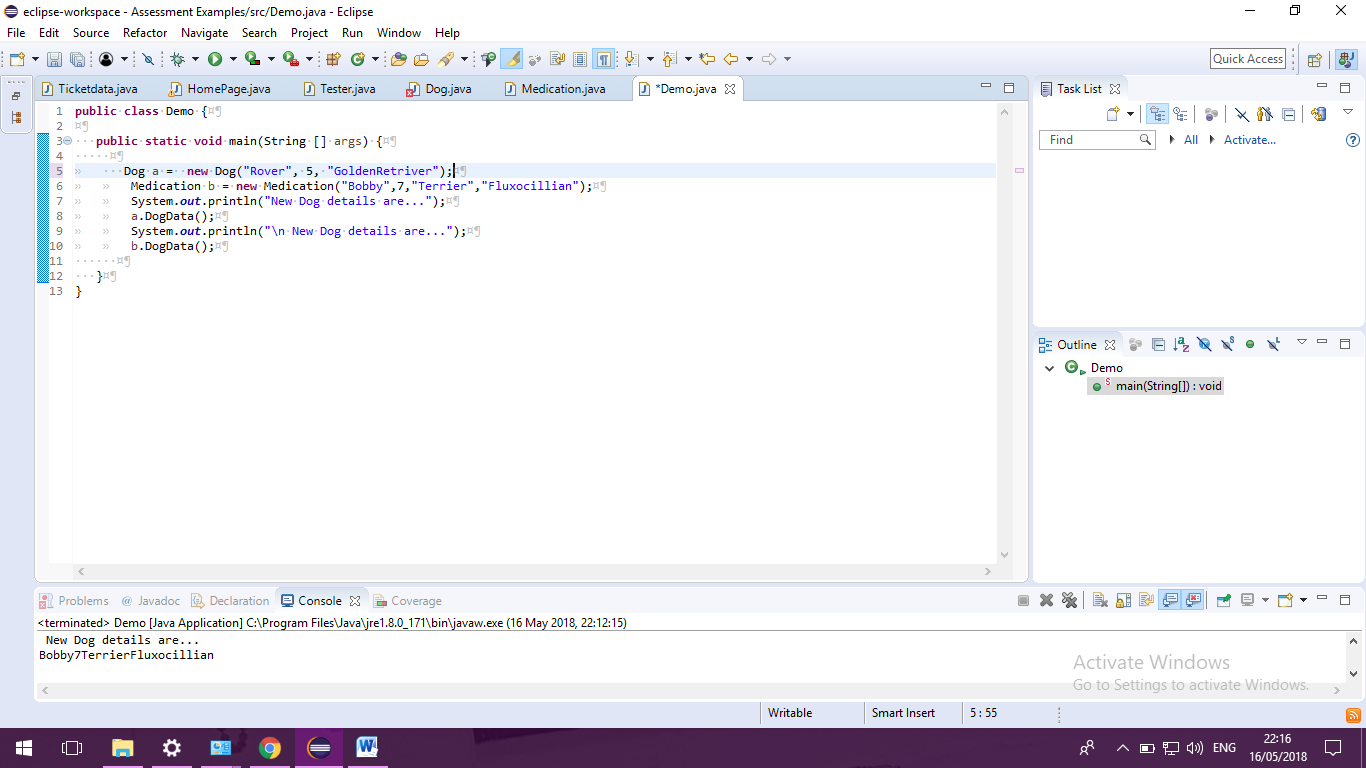
}

}



As you can see in the console above I have tried to run a method from the dog file with an abstract class and it has come up with the error cannot instantiate the type.





Once you remove the abstract class from the main class then you can use that class to instantiate methods and data from it.