**INTRODUCTION:**

This project is an attempt to demonstrate the use of object oriented programming concepts in Java such as Compile Time Polymorphism, RunTime Polymorphism, Data Encapsulation,Inheritance,Abstract classes and use of Classes and Object creation. The project consists of 7 classes which are namely,

1. - Star.java
2. - SuperGiant.java
3. - DwarfStar.java
4. - WhiteDwarf.java
5. - BlackDwarf.java
6. - BlackHole.java
7. - TestCases.java

The key goal is to establish various relationships between these classes although I have designed the project in such a way that not all are interdependent on each other.

Let us examine the classes step by step

1)- **Star.java and Supergiant.java:**

These two classes are connected to each other. These two classes can be an example to establish how **java establishes inheritance**. Our Star class is a base class which has certain properties namely name,constellation name,luminosity,temperature, flux and radius. It should be noted that all these variables are set as private and the only way to access and modify these variables is through getter and setter methods. This is a very good example of **Data Encapsulation** in Java. The Data and the Methods are kept apart so that no one can modify these properties very easily. We can also calculate the Flux of the Star with the help of an accessory method which is provided in the star class by providing some properties of the methods as input to the method. Now Supergiant is another class which inherits the Star class. This is an example of inheritance both here and in the real world. Technically Supergiants are stars but which are very large and certain properties. We can use **inheritance to model the real world relationship between stars and supergiants**. Supergiant class has one extra attribute which is surface gravity. Note that supergiant **has access to all the properties of its parent class star and since these properties are private, they can be accessed only by using get and set methods.** Besides, **all the properties of the star are inputted by using the scanner class demonstrating the use of I/O functions in Java.**

**So objectives demonstrated using these two classes are**

**- Use of Inheritance in Java.**

**- Demonstrated Encapsulation.**

**- Demonstrated using I/O using Scanner class.**

2)- **DwarfStar.java, WhiteDwarf.java and BlackDwarf.java**:

DwarfStar is another class which inherits the main Star class as its parent because Dwarf Stars are a special type of Stars. But this class is made **abstract** specifically to demonstrate how abstract classes can be used to provide just the **blueprint** or the implementation strategies for other methods. I have made the dwarf star class abstract because dwarf star can be further classified into many categories and I thought abstract classes would be a good design philosophy to use here where Dwarf Star class will just have some abstract methods which would later be implemented by its inheriting classes. Both BlackDwarf and WhiteDwarf which inherit the DwarfStar inherits its properties and abstract methods. **The implementation of the definition of the abstract methods of the DwarfStar is provided both in BlackDwarf and WhiteDwarf Star classes. This is a good example to demonstrate abstraction and abstract classes in Java.** The functionality which we can do with these classes is to calculate the pressure of these stars by inputting the star’s radius.Also, I have created a child class object with a parent class reference in the WhiteDwarf.java class to demonstrate virtual or run time polymorphism.This concept is known as method overriding and is an important concept of Object Oriented Programming.

Object Oriented Concepts used here:

* **Abstract classes and their implementation.**
* **Inheritance.**
* **Method Overriding or Runtime Polymorphism or Dynamic Dispatch**

**3)- BlackHole.java:**

This is an independent class which I designed to demonstrate the use of collections in java, overriding hashcode and equals methods which I think are quite important concepts in object oriented programming in Java. We also store all the BlackHole classes in a HashSet collection. We also create three constructors demonstrating default and parameterized constructors. **This is an example of Method Overloading which is compile time polymorphism a feature of Object Oriented Programming.** So, if we create a default object, the values provided in the default constructor would be assigned. The most important part of this class is when we create 2 objects with the same value and then add it into the HashSet. Since we override, HashSet and Equals() methods, the object gets added only once. If we do not override HashCode and Equals() method, the object gets added twice irrespective of being the same. The added functionality of this class is that we calculate the schwarzschild radius of the BlackHole by using its properties.

Demonstrated Principles in this Class

* **Method Overriding using Hashcode() and Equals() methods.**
* **Use of collections through sets (Equivalent of C++’s STL).**
* **Constructor Overloading which is compile time polymorphism in Java.**

**Building a Build System**

I did create my Java Project into a Maven Project but could not create a final build system because there is no singular objective/ functionality and the classes are quite independent of each other. But, I am aware of using Maven, Gradle or Ant for Java to create a build system.I will try to implement a project where all classes are interconnected with a single functionality next time and create a single build system.

**Writing Test Cases:**

I have tested all the functionalities(calculating Schwarzschild Radius, Testing Default BlackHole object creation, Calculating Pressure of the star) in the test cases class. I thought it would be a good idea to give the user some control here by inputting the necessary values and I have added various unit tests for testing all these functionalities. The testing framework which I have used for this Java Project is Junit.

**Documentation:**

This word document serves as documentation. I have also added comments extensively throughout the code to provide documentation. I have not used an auto code documentation tool since the documentation consistently changes with every code change I make.