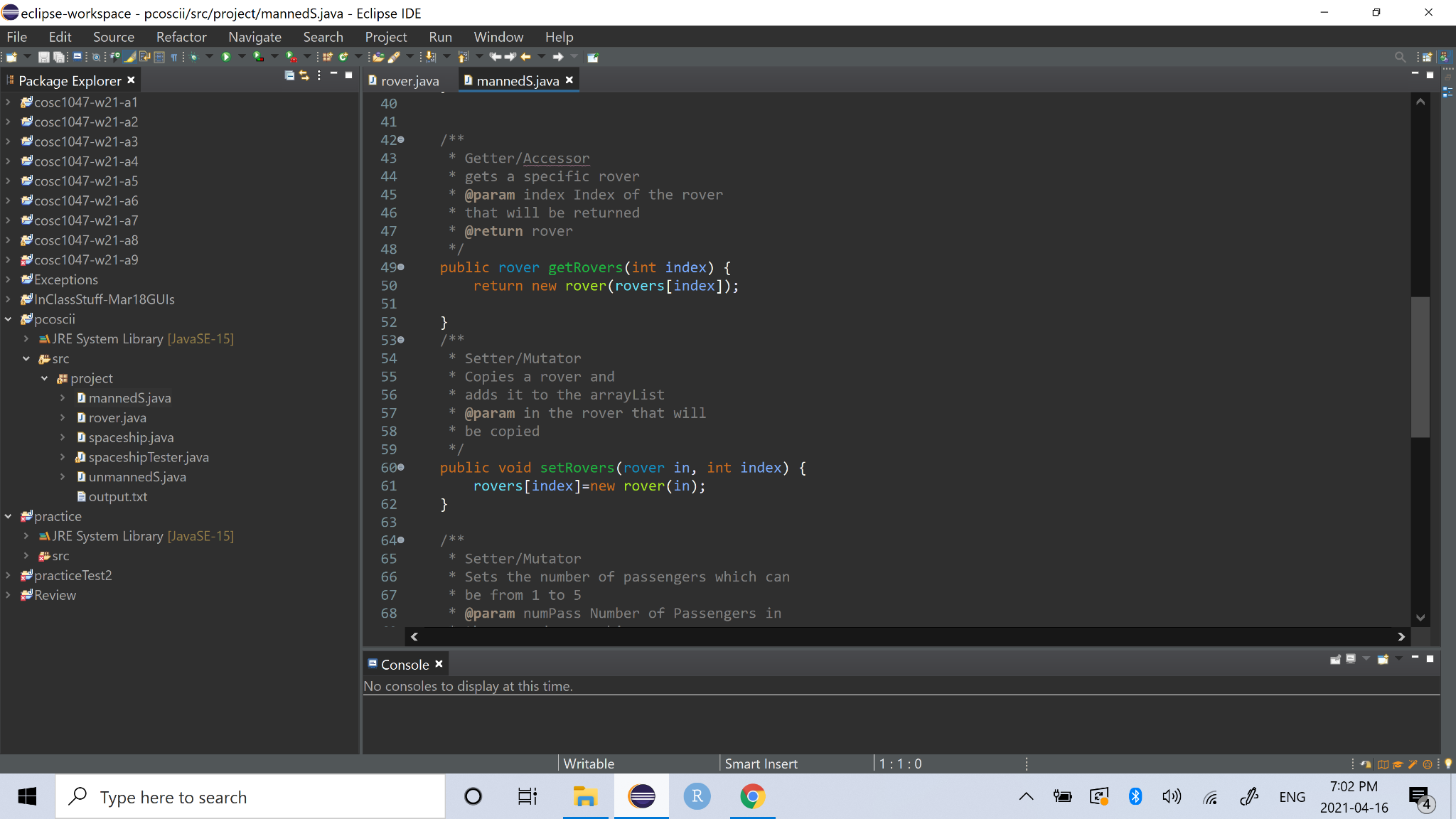
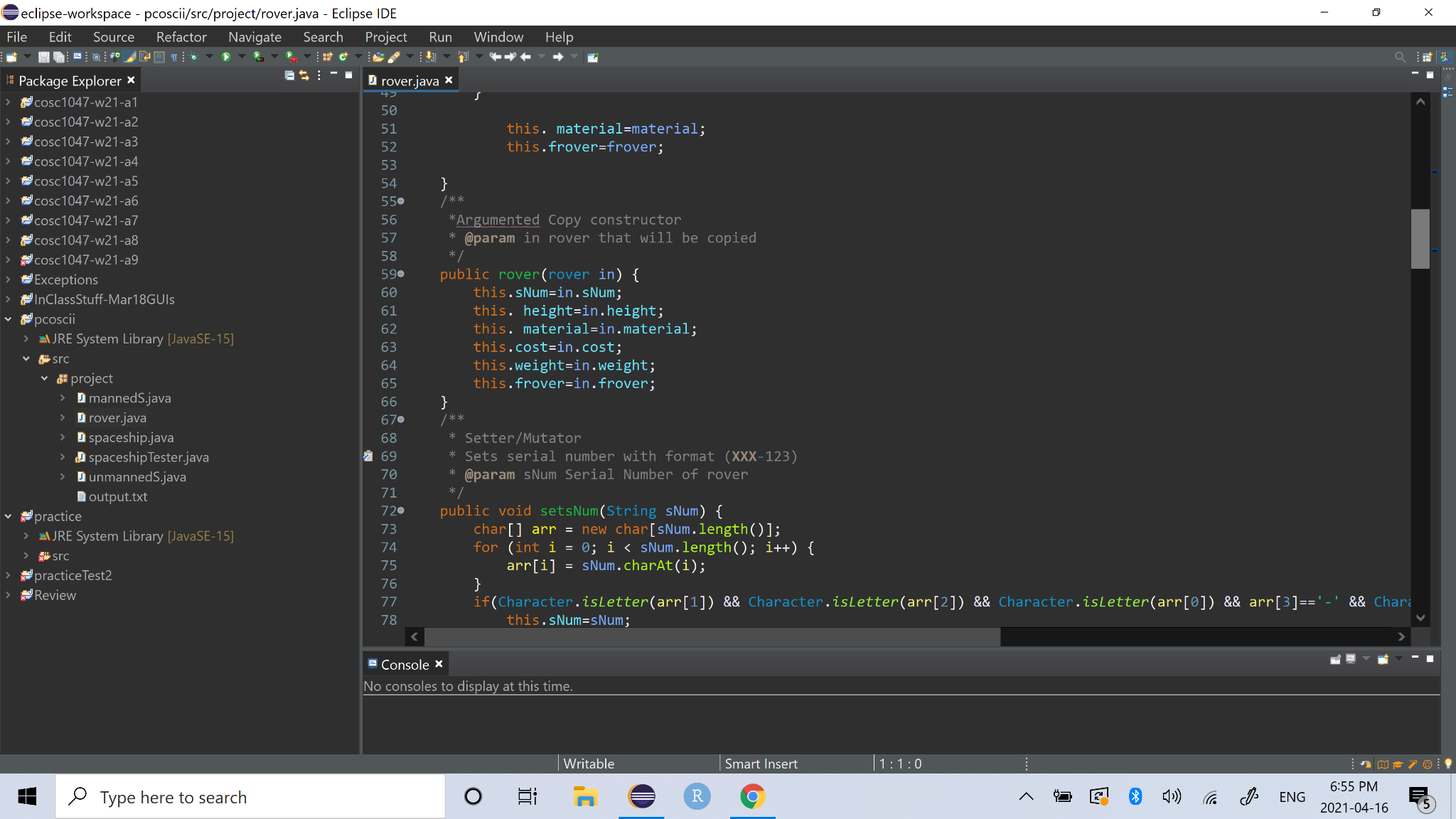
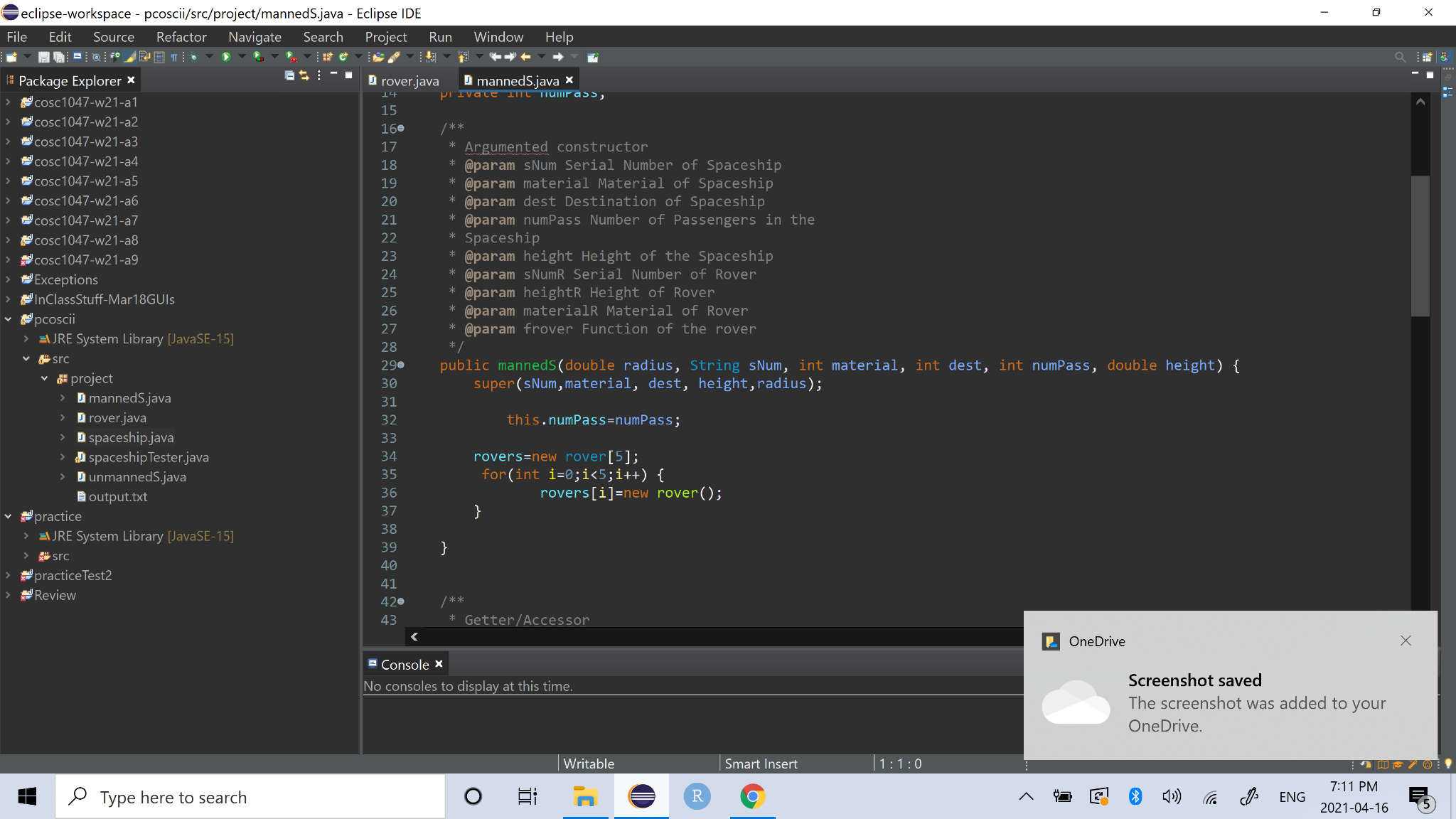
The Space Mission Builder (SMB) is a software application that will allow future astronauts to plan and design a space mission, the SMB builds Manned, Unmanned Spaceships and Rovers.

The goal of the SMB is to allow astronauts to gain more knowledge and experience regarding the process of planning and executing successful missions, it gives an insight into the intricate and detailed work a space mission entails and how a factor like cost can affect the viability of a mission.

The SMB implements Objects and Advanced objects because it is essential for the functioning of the software. The rover class is an advanced object because it allows us to have a copy constructor which takes a rover as an argument. The copy constructor is implemented in this code because it helps us create the rovers in the mannedS and unmannedS classes.



The setRovers method has two arguments, a rover and an index, it sets the rover in that index equal to the rover that is received in the method by using the copy constructor.

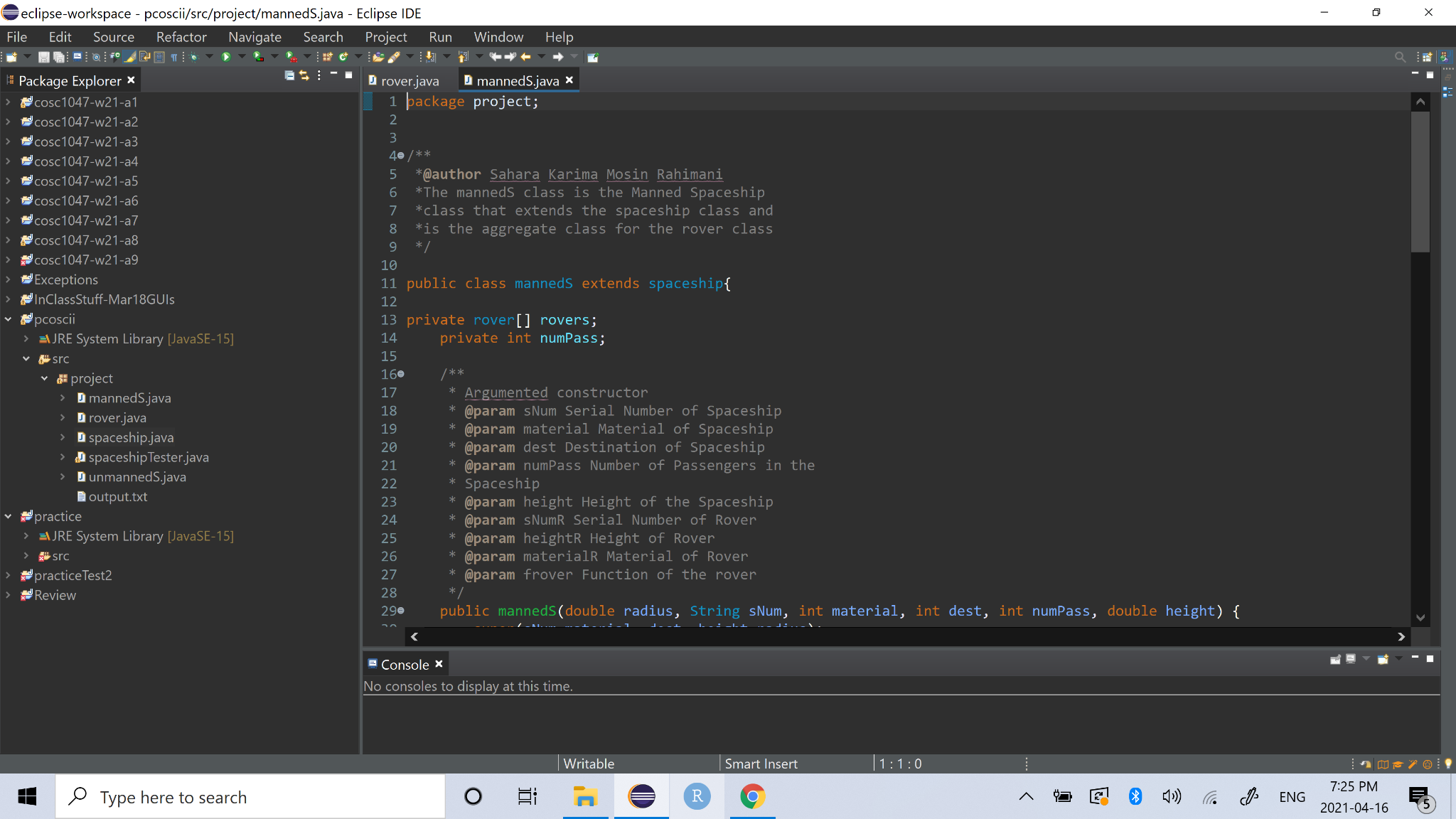
The SMB implements objects because it allows us to create highly detailed manned Spaceships and unmanned Spaceships, the objects contain multiple data fields that should be defined to get accurate statistics of the object. To create the object we use a constructor the mannedS constructor has the radius, serial number, material, destination, number of passengers and the height as arguments and it also sets the array of five rovers, which allows us to describe a manned spaceship as accurately as possible. 

polymorphism: allows us to use one command for all objects created

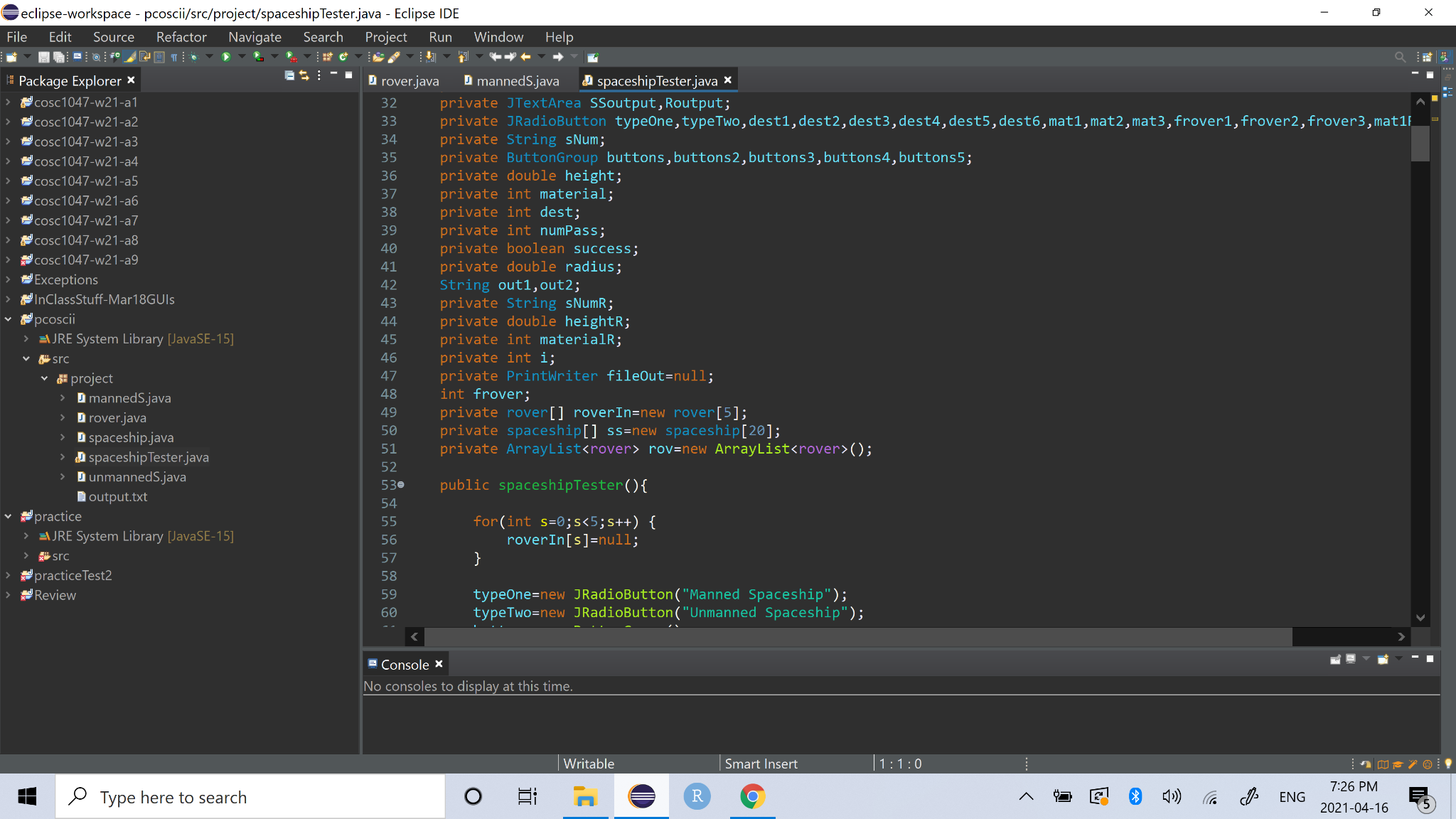
data leaks: aggregation allows us to have a relationship and the data leak fixes allows us to not lose information about spaceships and rovers.

The SMB implements inheritance in the mannedS and unmannedS classes as it allows it to create a specialized version of other objects which in this case is the spaceship object. Inheritance establishes a “is a” relationship between the objects, meaning that mannedS and unmannedS are spaceships. Inheritance allows the mannedS and unmannedS to access all the characteristics(data fields, accessors and mutators) of a spaceship plus specific characteristics that are present in their own classes, this allows us to avoid repetition of code and to reduce the amount of code used in the SMB.

The SMB also implements aggregation in the mannedS and unmannedS classes as it allows it to describe the mannedS and unmannedS objects as realistically as possible because in real life objects are made of other objects, which is exactly what we do when we use aggregation. Aggregation creates a “has a” relationship between unmannedS and mannedS and the rover. This means that the manned spaceship has a rover or the unmanned spaceship has a rover. With aggregation the rover in the unmannedS and mannedS classes has access to all methods in its class which allows us reduce the amount of code that we need to use and avoids repetition of code.

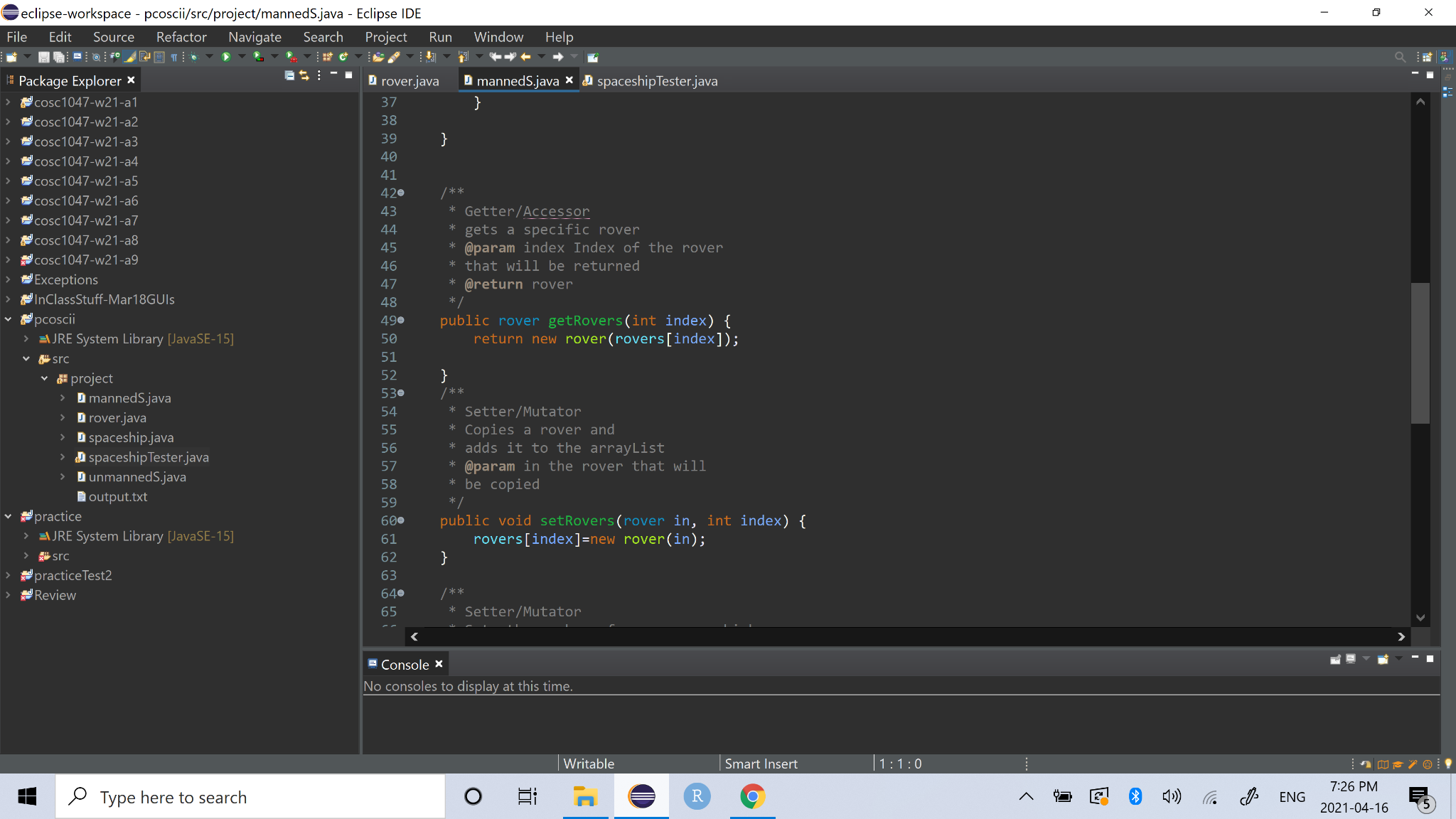


The SMB implements polymorphism as it allows us to avoid repetition of code. In the spaceshipTester we create a spaceship array that can hold both types of spaceships manned and unmanned. This makes it easier to store and keep track of all the spaceships regardless of the type.

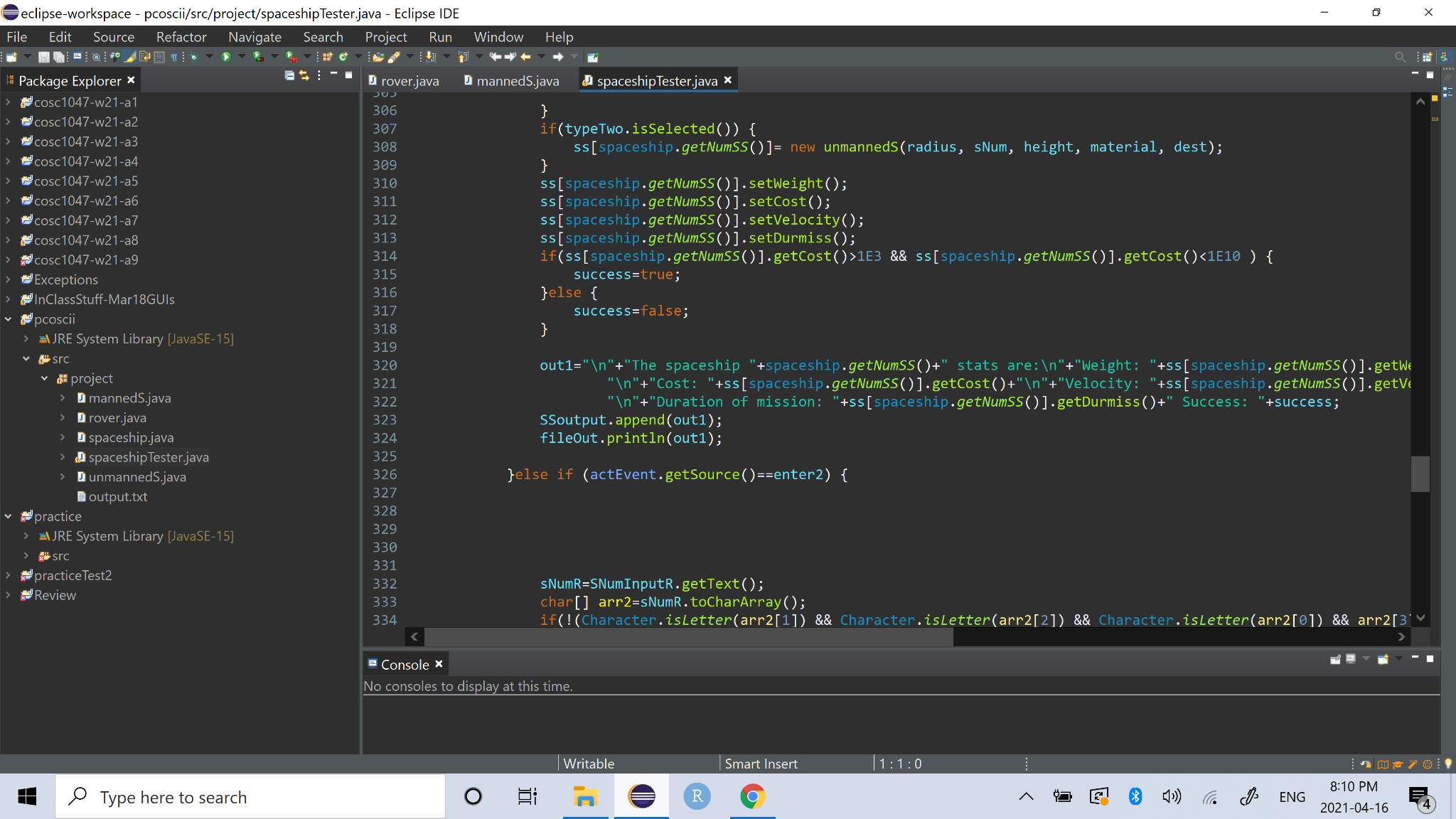


When we receive an aggregated object or when we return an aggregated object data leaks may occur if shallow copies and shallow returns are used, which means that we need to use deep copy and deep return.

Deep copies occur when you copy an object and also copy the object it references and when performing a deep return we return a reference to a copy of an object which is exactly what we do in the getRovers method.



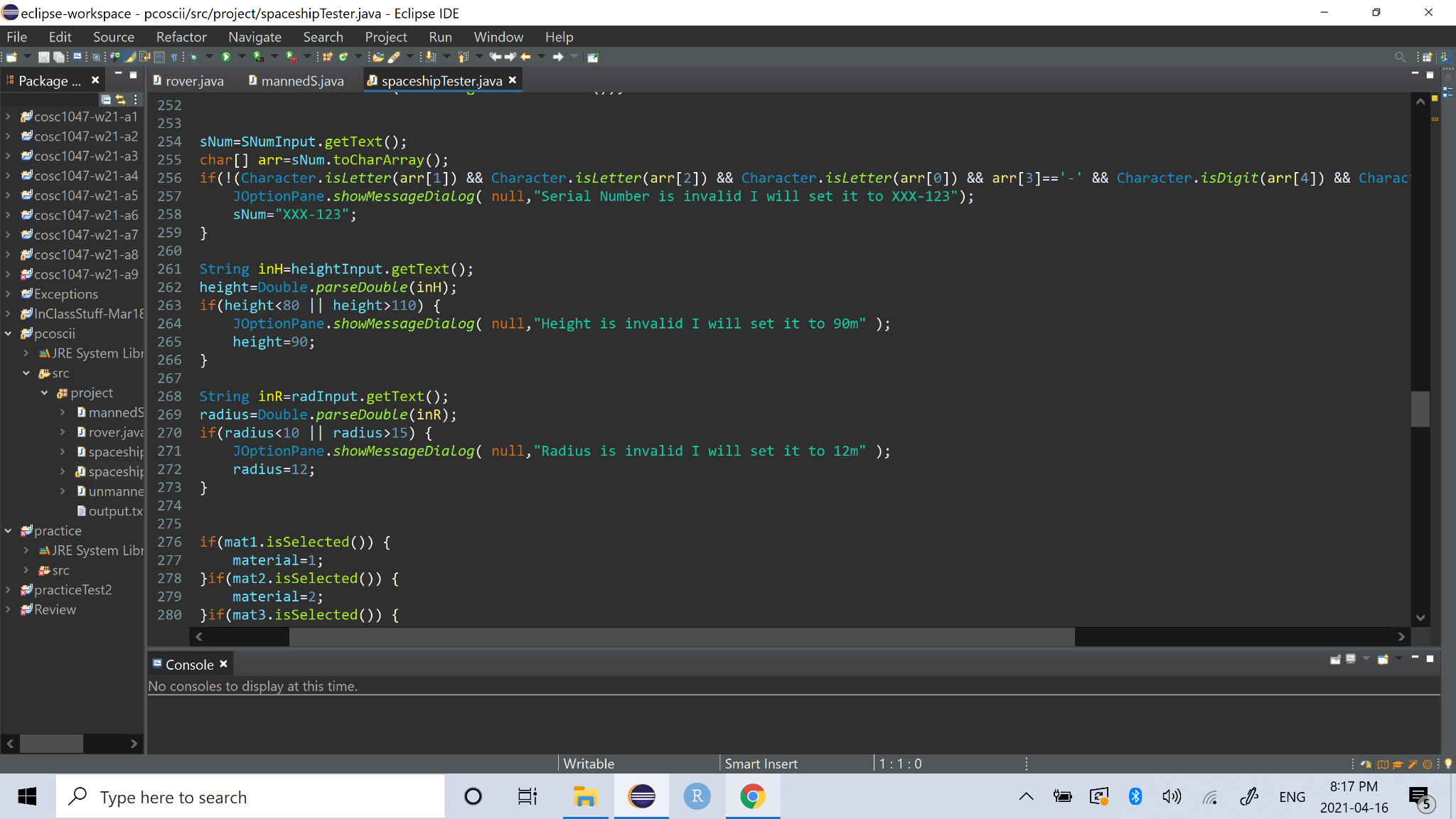
The SMB implements file output because in spaceflight it is essential to have documentation of the spaceship in case the user needs to perform a review of the statistics of the spaceship before proceeding to create other spaceships. So in the SMB the statistics of the latest spaceship is stored in the output.txt file. Which is what we see in the code snippet below.



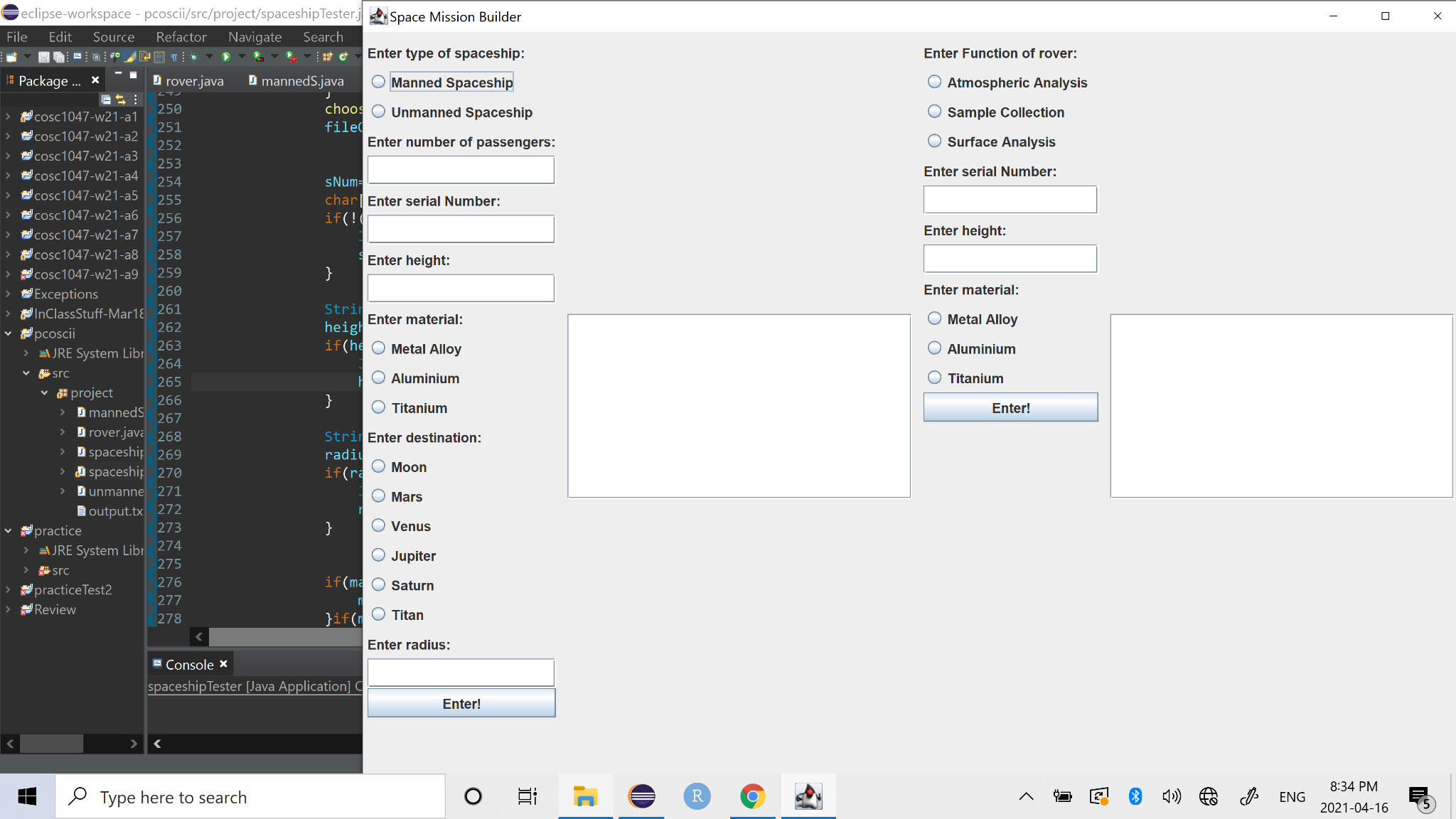
String processing is essential when we need to check the user input, especially when it comes to the serial number of the spaceships or the rovers. To make sure that serial numbers follow the format, each character will be checked using toCharArray, isDigit and isLetter. If all the conditions are not met then the SMB alerts the user using the JOptionPane that it will be assigned a default value because the format is not proper.

String processing is also used when collecting the data from the GUI because we need to use Double.parseDouble() and Integer.parseInt() to convert the strings to doubles or integers.

As shown in the example:



The SMB implements GUIs because it allows the user to interact with the code, it makes the format neater and easier to understand, the other reason why it is very useful is because it ensures that the user will only enter input with the correct format(when using radioButtons), it eliminates the possibility of having spelling mistakes and format mistakes, it is eliminates the necessity of a loop because the ActionListener generates a loop that is always listening for Events and it also allows us to see the statistics in the GUI using the TextArea .



The SMB implements ArrayLists because it will expand automatically when we add an object to it, we can easily do this by using add(). This is ideal because of the amount of rovers that are present in the spaceshipTester, with the ArrayList we can add all the rovers that we want without the need to know the size the final ArrayList should be. The ArrayList also allows us to easily get any rover that we want with get(index).

