1. The use of an array to manage reference to Event objects was helpful in creating this program. It enabled a numerous amount of events to be created without having to explicit call the object of the class ten times (and deal with all the variables separately); it stream-lined the process. This program would have been able to run and compile if I had created ten separate variables; however, it would have involved a lot more lines of code. This would have made the code harder to write out and harder to read-through.
2. In aggregation a class uses another class as a variable; they refer to this as a “has a” relationship. A College class would have a Class class variable. In composition a class is dependent on another class in order to exist. For example in this project OurDate is dependent on Event; if there is no Event; OurDate doesn’t exist. The relationship between Planner and Event is aggregation because Event is used inside the Planner class, however, Event class is not dependant on Planner for existence.
3. Separating the test classes from the program classes enables you to organise your files in the workspace. When a java project becomes more complex, there can be numerous classes and can become confusing. Separating the actual program from the test program is a good thing to do as they are technically two different programs. The Junit test is only for the programmer. If you were to send your java project to someone else; they technically don’t need the test class. Creating two different packages keeps things organised and enables you to only send your program, instead if your program and the test program.
4. Automated testing would definitely help when changes are made in the future to the program by ensuring the same outcome is being produced. The program is written and needs to meet certain parameters. If there wasn’t an automated testing procedure, you would have to go through the program and test it step-by-step and variable-by-variable in order to confirm it is running correctly; which could be a very time consuming procedure. Automated testing with Junit is a quick way to ensure any changes to the program don’t mess up the functionality and outcome of the program and ensure it’s still running correctly, with very little effort.
5. There are numerous other features that could have been tested using Junit in the OurDates class. You could test the setDay method as there are very specific restrictions based on which month you chose. Enabling a Junit class to test this would ensure your program only accepts days within the appropriate range. Testing setMonth would enable you to ensure a month is inputted within the 1-12 range. Another method that could be checked is the isEqual method; it could be tested to ensure the code is proper and is comparing two different dates for equality. The isLeapYear method is a very important one that could be tested in Junit. Calculating a leap year can be tricky as it doesn’t always happen every four years. The formula to calculate if a year is a leap year is a little complex. Testing this formula would be a good idea and would ensure you have the code working correctly. CalcDays method would be another good idea to test in Junit. Since there’s an expected outcome, and it also invokes the isLeapYear method; it would be a good idea to test this method in Junit to ensure it is calculating properly. Finally, the addOne method would be another method worth testing in Junit.