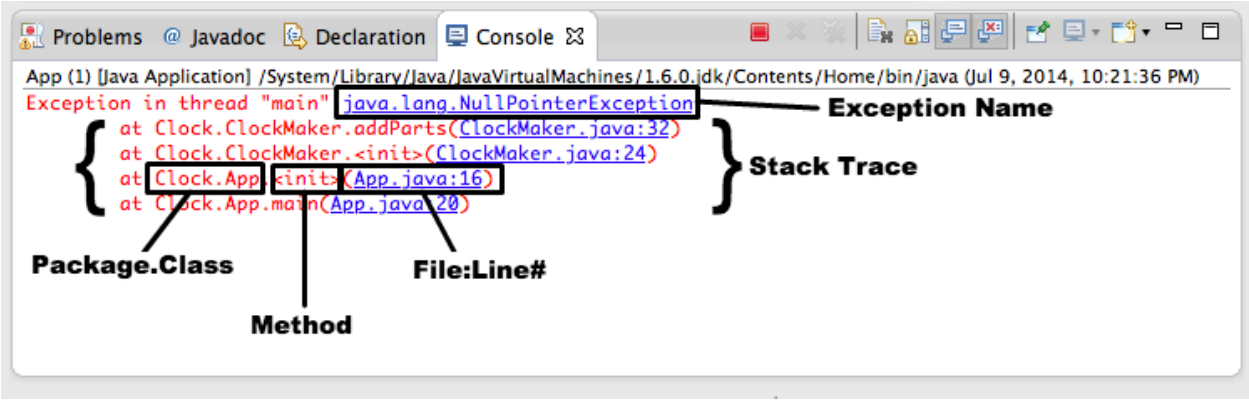
Runtime Errors

A runtime error occurs while your program is running. Sometimes these errors can be tough to track down, so here is a quick guide to assist in debugging runtime errors.

If your program crashes, Java will print the **stack trace** to your terminal. The stack trace is a list of all of the methods that were being executed when the program crashed. You'll find that the bug is almost always in one of those methods.

The topmost method is the method that got called most recently. Most likely, it's where the error happened. Moving down from there, you can trace the sequence of methods that were called to the first method that was called.

In a typical stack trace, there will be many methods that you did not write near the bottom of the list (they are either standard Java methods or CS015 support code methods) and your methods are usually near the top. *You can safely assume that the Java base code and the CS015 support code is bug free*, so you should begin tackling the stack trace when it first enters one of your methods.



1. **Stack Trace**: The list of all the methods executing when the error occurred. Start from the top and look for the first method that you wrote, as the error probably occurred there.
2. **Exception Name**: The error that occurred. This will usually give you an idea of what might have happened and how to fix it. (A little about Java terminology: when a runtime error occurs, it is described as Java "throwing an exception".)
3. **Package.Class**: The location where the error occurred.
4. **Method**: The method where the error occurred. **NOTE:** <init> refers to the constructor.
5. **File and Line #**: The name of the file and the line number where Java thinks the error occurred. It's a good idea to go to the line the compiler is referring to to begin debugging.

NOTE: The class or method that ran into the error is not necessarily the one that caused it. This happens most often if the method accepts a parameter. For example, if you passed a null value as the actual parameter into a method, an error would result because a

null value should never have been passed. You can, however, trace down from this class or method to figure out where your error was caused.

Here is a review of most common runtime errors.

Common Runtime Errors

NullPointerException:

Thrown when Java encounters a null reference when it doesn't expect one. This usually happens when you try to use something that hasn't been initialized or instantiated, or something that no longer exists because it was garbage collected or fundamentally altered.

Null pointers will occur most often when you create an instance variable and try to use it before you've actually assigned it a value. See below for an example of this occurrence.

```
public class OperateCar(){
      private Car _car;

      public OperateCar(){ }

      public void drive(){
            _car.moveForward(1);       //NullPointerException
            occurs
      }
}
```

The above code would be fixed by initializing `_car` in the constructor, which would look like this: `_car = new Car();`

ClassCastException:

Happens when your code attempts to wrongly cast an object to a particular class. Specifically, your object is not an instance of this class, or your object is not a subclass of this class. For example, if you try to cast an Car to a Color

```
Car myCar = new Car();
Color myColor = (Color) myCar; //generates ClassCastException
```

Arithmetic Exception:

Illegal arithmetic attempt. Most likely you tried to divide by zero.

StackOverflowException:

More methods got called than Java memory can handle. It is most likely you have an infinite loop or a problem with your base case in recursion, which you'll learn about a little later.

Note: Sometimes it may take a while before the StackOverflowException is actually thrown. If your program is taking a really long time to run, then you *might* be in the middle of an infinite loop.

```
Boolean alwaysTrue = true;
while(alwaysTrue){          //will run infinitely causing an
error
      myCar.drive();
}
```

IndexOutOfBoundsException:

When Java tries to access an Object in a list at an index that is not in the list's range.

```
Car[] listOfCars = new Car[5];  //array with indexes 0 through 4
for(int i  = 0; i <= 5; i++){
      listOfCars[i].drive();   //when i = 5, error will be
      thrown
}
```