1) What is your definition of an object?
An object is a data structure that includes both data as well as instructions to modify the data contained within the data structure(object).
2) What strategies should be deployed in terms of accepting input (i.e. the large number of objects.)?
Take in a list of objects as a command line parameter, or as a function parameter.
3) Is your sort generic? What sorting criterion does your system use? Is this criterion flexible; i.e. changeable by the user at the point of execution? What limitations have you placed upon legal sorting criteria? To make it reusable to other people, how should we include it into the Ruby Class hierarchy?
Our sort will be generic. The sorting criteria used will be that the class has to implement Comparable (or atleast the ⇔ operator) to specify how to sort similar objects. This will allow the user to specify their sorting criteria at the execution time.

One restriction is that all items in the list must be the same of the same class. Ideally we would like to support superclasses but for right now it is objects of the same type. Additionally the class must implement comparable.

4) In reality we have a uniprocessor system, describe "equationally" what is happening to your solution as you increase the concurrency (e.g. produce a regression model (remember regression from Statistics) of processing time against the number of threads. Your solution can be modeled as a program which: (1) has a component which produces threads; and (2) a set of threads which undertake the same (small) task. This is in essence the basis of stress testing (discussed in ECE 322)
Our solution will slow down the greater the number of threads there are. Each thread will have to alternate control of the CPU and as more threads are spawned, the CPU has to handle greater and greater loads. Context switching takes time, and the more threads there are, the more significant this context switching overhead becomes.

5) Concurrent systems tend to crash frequently – what approach to exception-handling have you devised? Consider the content of the library at: http://c2.com/cgi/wiki?ExceptionPatterns; which are applicable to this problem? Is Module Errno useful in this problem? What components of the Ruby exception hierarchy are applicable to this problem? Discuss in detail your strategy for exception-handling.

Our approach is to write the Exception handling code (try/catch block), at the highest level of our thread-based sort (ie, main thread). The child threads, that contribute to the sorting process, will throw custom exceptions with a message identifying thread-number if anything goes wrong. This way we can identify not only the exact reason why program encountered, fault, but also which thread caused it. Essentially, implementing the "NestedException" handling approach.

Errno is useful because that given we are working with threads, there are high probability of error and with the Errno module we can tell exactly what operating system errors occurred. It is more a convenience, rather than a necessity.

The components in the ruby exception hierarchy that are applicable to this problem mostly reside under StandardError, ie ThreadError, RuntimeError,IndexError...

6) What differences exist between thread-based and process-based solutions? How has this impacted the design of your solution?

First off threads and lighter than processes. Each thread shares a common heap but has a private stack. This has the consequence of threads having less overhead to setup and to communicate between each other.

However a process based solution is less error prone to crashing. If one thread crashes, the entire process will be taken down with it. However if only one process crashes, the other processes can continue their computation without issue and the master process can attempt to restart the crashed process. In the event the crashed portion is unable to be completed, the master process can then decide how to proceed (i.e. provide a partially sorted list or just display a general failure message).

7) Do you have any race-condition or task synchronization concerns about your solution? How do we tidy-up a multi-threaded program, if stopped mid-execution?

Yes there will be race conditions in our solution. Our solution will be spawning threads and race conditions arise when we have to join each thread back up to get the final sorted list.

8) As discussed in CMPUT 301: What is configuration management? What is version control? Are you using either concept? If "yes", describe your process and any tool support what you utilize – illustrate your process with regard to Assignments 1 and 2; if "no", justify your decision.

Configuration Management is a set of processes, policies, and tools that organizes the development process. It simultaneously maintains the current state of the software (called the "baseline"), while enabling developers to work on new versions for features or fixes. Version control is part of configuration management and is a way to organize and control revisions during development. We use github which is a versioning tool that allows us to work in our workspaces independently until we want to consolidate and merge our code. It allows us to also revert to previous working revisions if we run into issues with the code.

9) What is JRuby?

JRuby is an implementation of the Ruby programming language atop the Java Virtual Machine, written largely in Java.

10) Briefly Explain:

a. What is refactoring (as discussed in ECE 325 / CMPUT 301)?

Refactoring is modifying the design of the code to get rid of bad code smells. The functionality of the code should remain the same, but refactoring may allow for greater coherency to other people reading the code.

b. Are you using refactoring in your development process? Justify your answer?
We refactor very obvious code smells. For instance if there is a lot of duplicate code, we may apply the “Extract Method” technique to reduce duplication and increase readability. Otherwise, the best way to avoid having to refactor is to use good object-oriented design in the first place.

c. If “yes”, give examples, minimum of 2, of the refactoring “patterns” that you used in Assignment 1
We used “extract method” when initializing sparse matrices so that we did not have to duplicate code every time we needed to initialize a matrix. We also used the “introduce assertions” technique which introduced assert statements to only execute code when a condition was true. For instance, we can only find the inverse of a matrix if it is square. This technique directly overlaps with the use of the design-by-contract methodology

11. B. Explain in detail what happen when we execute the following code block

```
p 1.upto(Float::INFINITY).sloth
  .map { |x| x*x }
  .map { |x| x+1 }
  .take(5)
  .to_a
#=  [2, 5, 10, 17, 26]
```

Gets the first 5 values of x^2 + 1, starting at x = 1, and prints them. Starting at value x = 1, the first map squares the value and returns it. This returned value is then incremented by 1 from the second map. The take function returns the first 5 returned values and the to_a converts the values to an array.

12) a- Consider the following Java code segment, what design pattern is being implemented?

```
public class DesignPattern {
   private static DesignPattern pattern;
   private DesignPattern(){}
   public static DesignPattern getPattern(){
      if(pattern == null){
         pattern = new DesignPattern();
      }
      return pattern;
   }
}
```

It is implementing a Singleton Design Pattern.

b. The following code segment seeks to improve the above pattern. Fully explain what improvements are being made. Why do you think the code has two if statements?

```
public static NewDesignPattern getNewPattern(){
   if(pattern == null){
      synchronized (NewDesignPattern.class) {
         if(pattern == null){
            pattern = new NewDesignPattern();
         }
      }
   }
   return pattern;
}
```

The addition of Synchronized ensures that when accessing the NewDesignPattern.class, it is done atomically such that the process is thread safe. The reason why there is an outer If statement is because we will only ever make 1 instance, so if it does not have outer if statement, then every call to getNewPattern will have to go through Synchronized, which is expensive. Instead, add an outer if statement such that when an instance is created, we no longer have to go to synchronized.

c- How would you implement this improved pattern in Ruby?

```ruby
class Configuration
  # the class variable @@instance holds
  # a reference to the singleton object. These are shared between all
  @@instance = 0
  @@semaphore = Mutex.new

  # the class #instance method returns the
  # single reference to the object instance.
  def self.instance
    if @@instance == 0
      @@semaphore.synchronize do
        if @@instance == 0
          @@instance = Configuration.new
        end
      end
    end
    @@instance
  end

  # The #new method is privatized, disallowing
  # the creation of additional instances of the
  # Configuration class, and limiting instances
  # of the Configuration class to the @@instance
  # variable provided by the #instance method.
  private_class_method :new
end
```

d- What is the difference in thread-models, between Java and Ruby?

The main difference between ruby threads and java threads is that ruby threads uses green threads where the interpreter implements them, while Java threads uses native threads that is created by os
This means that can only use 1 core and 1 cpu despite maybe having multiple cores.
Advantage with green threads however is that multithreading will be available always, even in systems that do not support threading.

*although real threads are implemented in current versions of ruby?

From https://www.ruby-forum.com/t/ruby-versus-java-threading/108002/2

e- What is the difference in thread-models, between Ruby and JRuby?

Since Jruby is ruby implementation in java, Jruby real threads schedules by the OS. This means that it can utilize multiple cores and therefore will run faster on multicore system compared to normal ruby threads.

13) Consider the following Java code segment:
Shares.java

```
public static final List<String> symbols=
Arrays.asList("IBM","AAPL","AMZN","CSCO","SNE","GOOG","MSFT","ORCL","FB","VRSN");
```

GoogleFinance.java

```
public static BigDecimal getPrice(final String symbol) {
  try {
    URL url=new URL("https://finance.google.ca/finance?q="+symbol);
    final BufferedReader reader= new BufferedReader(new InputStreamReader(url.openStream()));
    final String data=reader.lines().skip(170).findFirst().get();
    final String[] dataItems=data.split(">");
    return new BigDecimal( NumberFormat.getNumberInstance().parse(dataItems[dataItems.length-
1].split("<")[0]).toString());
  }catch (Exception ex) {
    throw new RuntimeException(ex);
  }
}
```

ShareInfo.java

```
public class ShareInfo {
  public final String symbol;
  public final BigDecimal price;

  public ShareInfo (final String theSymbol, final BigDecimal thePrice) {
    symbol = theSymbol;
    price = thePrice;
  }
  public String toString() {
    return String.format("symbol: %s price: %g", symbol, price);
  }
}
```

ShareUtil.java

```
public class ShareUtil {
  public static ShareInfo getPrice(final String symbol) {
    return new ShareInfo (symbol, GoogleFinance.getPrice(symbol));
   }
  public static Predicate<ShareInfo> isPriceLessThan(final int price) {
   return shareInfo -> shareInfo.price.compareTo(BigDecimal.valueOf(price)) < 0;
  }

  public static ShareInfo pickHigh(final ShareInfo share1, final ShareInfo share2) {
    return share1.price.compareTo(share2.price) > 0 ? share1: share2;
  }
}
```

PickShareImperative.java

```
  ShareInfo highPriced = new ShareInfo ("", BigDecimal.ZERO);
  final Predicate isPriceLessThan500 = ShareUtil.isPriceLessThan(500);
  for(String symbol : Shares.symbols) {
    ShareInfo shareInfo = ShareUtil.getPrice(symbol);
    if(isPriceLessThan500.test(shareInfo)) highPriced = ShareUtil.pickHigh(highPriced, shareInfo);
  }
  System.out.println("High priced under $500 is " + highPriced);
```

The previous code simply compares some shares' prices and returns the stock with the highest price whose value is less than 500$. However, the code uses an imperative style. As you can see, the code uses mutating variables. Furthermore, if we want to change the logic a little (i.e. pick the share with the highest price under $1,000), we will have to modify this code. This makes the code not reusable.

a- Rewrite the previous code in a functional style. You should create a class npftamed PickShareFunctional in which you will define one method called findHighPrices. To help you, this is how you should call findHighPrices method:

```
findHighPriced(Shares.symbols.stream());
```

As you can see, converting symbols from List to a Stream of symbols will allow you to use the JDK specialised functional-style methods (i.e map, reduce, and filter). Hint: In this method, you should do the following steps:
1- Create a list of ShareInfo filled with the price for each of the symbols in Shares

2- Trim down this list to a list of shares whose prices under $500 3- Return the highest-priced share.
Think Functional!

```
import java.math.BigDecimal;
import java.util.Comparator;
import java.util.stream.Stream;


public class PickShareFunctional {

    private ShareInfo base;
    BigDecimal limitPrice;

    public PickShareFunctional(BigDecimal limitPrice) {
        this.limitPrice = limitPrice;
        base = new ShareInfo("", new BigDecimal(-1));
    }

    public ShareInfo findHighPriced(Stream<ShareInfo> shareStream) {
        return shareStream.filter(x -> (x.price.compareTo(limitPrice) < 0))
                .reduce(base, (a, b) -> (a.price.compareTo(b.price) < 0? b:a ));
    }


}
```

b- Calculate the execution time for the findHighPriced method. i.e try to use timing methods in java before your call to findHighPriced methods. What is the execution time for the findHighPriced method? Out of the three steps explained above in part (a), which step do you think is responsible for most of this time?

The execution time for the stream method is 30636239 ns. I think the step that takes the longest time is step2

c- Change the way you are calling the findHighPriced method as:

```
findHighPriced(Shares.symbols.parallelStream());
```

```java
import java.math.BigDecimal;
import java.util.ArrayList;

public class Share {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        PickShareFunctional pickShareFunctional = new PickShareFunctional(new BigDecimal(500));

        long t1 = System.nanoTime();


        ArrayList<ShareInfo> shares = new ArrayList<>();
        shares.add(new ShareInfo("APPL", new BigDecimal(178.9)));
        shares.add(new ShareInfo("GOOG", new BigDecimal(1175.76)));
        shares.add(new ShareInfo("TSLA", new BigDecimal(290.92)));
        shares.add(new ShareInfo("NTES", new BigDecimal(232.95)));
        shares.add(new ShareInfo("NFLX", new BigDecimal(358.86)));
        shares.add(new ShareInfo("SOGO", new BigDecimal(6.34)));

        ShareInfo max = pickShareFunctional.findHighPriced(shares.stream());

        long t2 = System.nanoTime();

        // normal   28654050
        // parallel 34821759
        System.out.println(max.symbol);
        System.out.println(t2-t1);


        long t3 = System.nanoTime();

        ArrayList<ShareInfo> shares_2 = new ArrayList<>();
        shares_2.add(new ShareInfo("APPL", new BigDecimal(178.9)));
        shares_2.add(new ShareInfo("GOOG", new BigDecimal(1175.76)));
        shares_2.add(new ShareInfo("TSLA", new BigDecimal(290.92)));
        shares_2.add(new ShareInfo("NTES", new BigDecimal(232.95)));
        shares_2.add(new ShareInfo("NFLX", new BigDecimal(358.86)));
        shares_2.add(new ShareInfo("SOGO", new BigDecimal(6.34)));

        ShareInfo max_2 = pickShareFunctional.findHighPriced(shares_2.parallelStream());

        long t4 = System.nanoTime();

        // normal   28654050
        // parallel 34821759
        System.out.println(max_2.symbol);
        System.out.println(t4-t3);


    }

}
```

14) Create a class RLisp which presents Ruby implementation of the Lisp language. Lisp is a family of computer programming languages with a fully parenthesized prefix notation.

See rlisp.rb file.