Mock Term Test NWEN303 2021

Total 100 points

Duration: 2 hours (the lecture slot is 1 hour, so you can finish it at home)

Answer in the white space after each sub question. Each sub question gives you a full page to answer, but you are not expected to fill up all the space. Many questions have elegant short answers.

Write your name on the top right corner of all the pages.

Question 1: [total 40 marks]
Designing/implementing parallel algorithms in Java

PerfectImage is company producing an image processing tool, they have many filters, all implementing the filter interface:

```
public interface Filter{
  void applyFilter(Image img);
  }
```

A Filter takes an image and does some in place transformation on the image.
This means that applyFilter modifies the (large) Image object in order to obtain a filtered version of the same image.
Is it possible to set up a filter chain in this simple way:

```
void applyFilters(List<Filter>filters,Image img){
  for(Filter f:filters){
    f.applyFilter(img);
    }
  }
```

Now PerfectImage wants to support movies. In this simple context movies are just collections of images.
A naive attempt of applying all the filters to all the images works correctly, but is too slow.

```
void applyAllFilters0(List<Filter>filters,List<Image> movie){
  for(Image i:movie){
    for(Filter f:filters){
      f.applyFilter(i);
      }
    }
  }
```

Question 1a: [6]
They asked to their programmer Bob to write a parallel version for such task.
This is the first attempt that Bob comes out with:

```
private static final ExecutorService pool=Executors.newCachedThreadPool();
void applyFilters1(List<Filter>filters,List<Image> movie)
throws InterruptedException, ExecutionException{
  for(Image i:movie){
    for(Filter f:filters){
      pool.submit(()->f.applyFilter(i));
      }
    }
  }
```

With great happiness of Bob, this method executes very fast!
However, when Bob run the test suite on this method, sometimes the test fails
stating that some filters was not
applied at all.
Describe what happens in this code, and explain where is the error.

Question 1b: [6]
After understanding his mistake, Bob modify his code as follows.
The only code difference is in the lines marked with //#

```
private static final ExecutorService pool=Executors.newCachedThreadPool();
void applyFilters2(List<Filter>filters,List<Image> movie)
throws InterruptedException, ExecutionException{
  for(Image i:movie){
    List<Future<?>> results=new ArrayList<>();//#
    for(Filter f:filters){
      results.add(pool.submit(()->f.applyFilter(i));//#
      }
    for(Future<?> r:results){r.get();}//#
    }
  }
```

Now, when Bob, runs the tests, he discovers that some images get corrupted. He is very confused.
Describe what happens this time:
-Explain why some images are corrupted.
-State the name for this kind of problems.

Question 1c: [8]
Depressed but still not ready to surrender, Bob learn about the synchronized statement on a web forum and naively tries to use it.
The only difference is in the line marked with //#

```
private static final ExecutorService pool=Executors.newCachedThreadPool();
void applyFilters2(List<Filter>filters,List<Image> movie)
throws InterruptedException, ExecutionException{
  for(Image i:movie){
    List<Future<?>> results=new ArrayList<>();
    for(Filter f:filters){
      results.add(pool.submit(()->{ synchronized(i){f.applyFilter(i);} });//#
      }
    for(Future<?> r:results){r.get();}
    }
  }
```

Now, when Bob, runs the tests, they pass just fine!
Bob is very happy and goes to sleep.
However, the day after while doing some performance test he discovers that his code was much slower than the
sequential version, and only one core is ever used.
Moreover, sometimes the result is still different from what is expected!
Explain why:
-Only one core is used.
-It is even slower than the sequential version.
-The result could still be different with respect to the sequential version
-[HARD:] How could Bob improve his tests to automatically detect this change in behavior?

Question 1d: [20]

At this point Bob gives up and quit, promising that will never use parallelism ever again.

Now write your own version that is both correct and efficient.

Be careful, there may be other issues in Bob code that have not be underlined in the problems encountered before.

If you need any reasonable assumption about the shape of the object graphs, list them explicitly.

_____

Question 2: [40 total]

Fixing old and broken parallel Java code.

We have a Product class with a description and some review.
Method polite checks if the text contains any invalid idioms/phrases, and fixGrammar uses some AI to fix the grammar of the text. fixGrammar is guaranteed to not introduce any invalid idioms/phrases.
A polite Product is a product where both the description and all the reviews are polite.
Class ComplexCode contains method fix, that is HORRIBLY coded. I tried to insert as many anti-patterns and wrong
ways of coding and thinking as I could into it. The method fix is also pointlessly very long. It is “sort of correct”, butdoes have at least 2 bugs.
This question has 3 sub questions:
-Identify at least 2 actual bugs in the fix method.
-In addition to the actual bugs above, can you identify at least 5 or 6 ways this code is unprofessional?
-Write a very compact and correct version of the fix method. My model solution is 6 lines (plus header).

```
class Product{
  String description;
  List<String> reviews;
  Product(String description,List<String> reviews){
    this.description=description;
    this.reviews=reviews;
    }
  }
class ComplexCode {
  public boolean polite(String s){/*..*/}
  public String fixGrammar(String s){/*..*/}
  //the grammar of a polite Product is fixed, and true is returned.
  //if the product is not polite, false is returned and the product is not modified.
  public boolean fix(Product productId){
    String description;
    ArrayList<String> reviews=new ArrayList<String>(productId.reviews);
    String tag=new String("###");//created with new, have unique identity!
    description=productId.description;
    ArrayList<Thread> kools=new ArrayList<Thread>();
    for(int i=0;i<productId.reviews.size();i++){
      final int j=i;
      Thread koolParrallel=new Thread(){
        public void run(){
          String current=reviews.get(j);
          if(polite(current)){reviews.set(j,fixGrammar(current));}
          else{reviews.set(j,tag);}
          }
        };
      koolParrallel.start();
      kools.add(koolParrallel);
      }
    if(polite(description)){
      description=fixGrammar(description);
      }
    else{
      description=tag;
      }
    if(description!=tag){
      for(Thread t:kools){
        try {
          t.join();
          }
        catch (InterruptedException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
          }
        }
      boolean hasTag=false;
      for(int i=0;i<productId.reviews.size();i++){
        if(reviews.get(i)==tag){hasTag=true;break;}
        }
      if(!hasTag){
        productId.description=description;
        productId.reviews.clear();
        productId.reviews.addAll(reviews);
        }
      else{return false;}
      }
    else{//we did not touch the Product!
      return false;
      }
    return true;
    }
  }
```

Question 2a: [10]
-Identify at least 2 actual bugs in the fix method.
[HINT: do not lose too much time if you can not find them]

Question 2b: [10]
-In addition to the actual bugs above, can you identify at least 5 or 6 ways this code is unprofessional?

Question 2c: [20]
-Write a very compact and correct version of the fix method.
My model solution is 6 lines (plus header).

Question 3a: [10]
Explain why so many methods in Java throws InterruptedException?
What is the purpose of this exception?

Question 3b: [10]
Consider the following code:

```
public String guessPassword(){
  while(true){
    String result=guessOneTime();
    if(result!=null){return result;}
    }
  }
```

Doing minimal code modifications, turn guessPassword into an interruptible method.