# MPP Final Exam 4/21/16

Name:_____ StudentId:_____

There are a total of 45points possible. Note there are questions on the back of this page. You have until12:00 Noon to complete all the questions.

---

1) We have a superclass Employee and a class Manager that extends Employee. The Manager class has a computeBonus method that is not in Employee.

   We decide to add a new class that extends Employee called SeniorTeamLeader. The SeniorTeamLeader computeBonus is identical to Managers. Using new Java 8 Interface features modify Manager and SeniorTeamLeader so that the computeBonus() code is not duplicated. Manager and SeniorTeamLeader still remain subclasses of Employee. Do not change the Employee class.

   A. (4pts) Show your UML class diagram for your solution.

      See diagram UML-prob1

   B. (6pts) Show code needed to implement the class diagram as follows –

      a. class declaration for Manager

```
public class Manager extends Employee implements HigherPay {
```

      b. class declaration for SeniorTeamLeader

```
public class SeniorTeamLeader extends Employee implements
HigherPay
```

      c. all code for any new item you added in the class diagram.

```
public interface HigherPay {
          default double computeBonus() {
// implementation here to match what was in Manager } }
```

      d. Describe any changes you made to the existing Manager class.

         Remove computeBonus() from Manager

2) (4pts) If we attempt to run the following code what will be the result? What is an appropriate fix for this code, and what will be the result from the fixed code?

```
Stream<Integer> stream2 =
                    Stream.iterate(1, n -> n + 1);
```

stream2.collect(Collectors.toList()).forEach(System.out::println);

This is an infinite stream so we will never get the print out – eventually it will time out.

Put a limit in stream2.collect.limit(100) to get 100 numbers starting from 1.

3) (8pts) We have a list of undergraduate students. Students have the following attributes:

```
double gpa;
String major;
```

Write a streams and lambda implementation that will generate a list of students called honorRoll that includes all the students that have a gpa greater than 3.0 and have a major of compSci.

```
List<Student> honorRoll = underGrads.stream()
                              .filter(s -> (s.getGpa() > 3.0))
                            .filter(s -> (s.getMajor() == "compSci"))
                        .collect(Collectors.toList());
```

4) (6pts)Suppose we want to sort the honoRoll list by gpa with highest gpa first. Show the changes you need to make to your lambda implementation and to the Student class for that case.

We can add the following prior to .collect in the answer above:

```
.sorted(Comparator.comparing(Student::getGpa).reversed())
```

Or we could create a StudentGradeComparator to use like this:

```
.sorted(StudentGradeComparator).reverse()
```

public class StudentGradeComparator implements Comparator<Student> {

```
@Override
      public int compare(Student s1, Student s2) {
            if(s1.getGpa() == s2.getGpa()) return 0;
            else if(s1.getGpa() < s2.getGpa()) return -1;
            else return 1;}
```

5) (4pts) We discussed two approaches unit testing stream expressions in class. Describe those two approaches. Explain which approach would be best for testing your answer to problem 3

a. If the pipeline is simple enough, we can name it as an expression and unit test it directly. In this case our pipeline is simple so we would use this approach.

b. For the complex approach to testing we replace a lambda that needs to be tested with a method reference plus an auxiliary method. Then you can test the auxiliary method.

6) For the following interface:

Public interface Triplet<S, T, U> {

public S getFirst();

public T getSecond();

public U getThird(); }

a. (5pts) Write a parameterized type implementation

```
public class ParamTriplets implements Triplet <String, String, Integer>{

	@Override
	public String getFirst() {
		// do some logic here
		return “first”;
	}

	@Override
	public String getSecond() {
		// do some logic here
		return “second”;
	}

	@Override
	public Integer getThird() {
		// do some logic here
		return 3;
	}}
```

b. (5pts) Write a generic class implementation

```
public class GenericTriplet<S, T, U> implements Triplet<S,T,U>{
 private S first;
 private T second;
 private U third;

 public GenericTriplet(S first, T second, U third) {
		this.first = first;
		this.second = second;
```

```
        this.third = third;
    }
        @Override
        public S getFirst() {
            // some logic here
            return null;
        }

        @Override
        public T getSecond() {
            // some logic here
            return null;
        }

        @Override
        public U getThird() {
            // some logic here
            return null;
        }}
```

7) (3pts) Employees is the superclass for SeniorTeamLeader subclass. Will the following code to compile? If not what change do we need to make to get it to compile?

```
List<SeniorTeamLeader>  list1 =  //populate with SeniorTeamLeader
List<Employee> list2 = list1;
```

*No it will not compile. Use the extends bounded wildcard to solve the problem*

*List<? extends Employee> list2 = list1;*