**Final Exam Notes**

**Benefit of Functional Programming:**

1) Functional programming does not cause any “side-effect” on parameter passed, which mean it perform some services/task and return back the result to caller.

2) Another thing related to functional programming is it has only output dependency as argument,

and this needs to be determined before output is generated.

3) Whereas in imperative we perform day to day duty, perform some calculation on certain variables, change it state and then produce the results. The example is looping statements, if-then-else clauses etc.

**Declarative Programming and Functional Programing**

“Imperative programming is like “How” you do something and declarative programming is more like “What” you do. Declarative programming is when you write your code in such a way that it describes what you want to do, and not how you want to do it. It is left up to the compiler to figure out the how. Declarative programming just passes the input and expects the output without stating the procedure how it is done. The one I am confused about is Functional Programming. I know Functional Programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data and is not a type of declarative language

Functional programming is that you want to achieve the result without modifying the existing value of certain parameter, this is the beauty of functional programming where the data comes as value and then perform some task and return the value back to caller. Such as Math.pow(x,y), this power function take two arguments, and return the value of x power y, without modifying any values of x or y.

**Explain the difference between *functional interface, functor,* and *closure*, and give examples of each using Java 7 syntax**

1) Functional interface: is Runnable interface like Callable, Comparator interface and a whole host of other interfaces defined by Java.

2) Functor: it requires to name your functional interface, and java always require such objects, this is called functor.it is an implementation of functional interface.

3) Closuer: the major buzz of Java is Lambdas and it is called closures, its function which refers to free variables that used locally. The variables that are not assigned to are closed over the surrounding state in order to bind them to a value. A closure is a functor embedded inside another class, that is capable of remembering the state of its enclosing object.

**Name three benefits of including functional style programming in Java**

1. It reduce creating different classes, you can embed this within your code

2. Reduce the writing long code, generate your results and then use it with different occasion.

3. More succinct and clear than anonymous inner classes. More concise coding. it is very easy for compiler to compile the program. The other major benefit is concurrency

**Question Section**

1) (8pts) We have a list called allUserStories that contains all the user stories that we have for a product. Write a streams and lambda implementation that will create a list called topUserStories that includes all the UserStories with priority set to 1 that are marked as not implemented. In addition to your solution, show what methods you need to have in the UserStory class to make your streams solution work.

*In UserStory add: public int getPriority() and public boolean isImplemented()*

List<UserStory> topUserStories = *allUserStories*.stream()

.filter(u ->u.getPriority() == 1)

.filter(u ->(!u.isImplemented()))

.collect(Collectors.*toList*());

2) (6pts) Modify your solution for problem 1 to produce a list of user stories called leastDevEffort that includes the 10 User Stories with the least amount of development effort that are priority 1 and that are not implemented. This list should have the User Stories with the lowest amount of development effort first. Show what methods you need to add in the UserStory class to make this new streams solution work.

*In UserStory add: public int getDevEffort()*

List<UserStory> leastDevEffort = *allUserStories*.stream()

.filter(u ->(u.getPriority() == 1))

.filter(u ->(!u.isImplemented()))

.sorted(Comparator.*comparing*(UserStory::getDevEffort)).limit(10)

.collect(Collectors.*toList*());

3) (10pts) Turn the stream pipeline solution for problem 1 into a reusable lambda library element.

Explain what the steps are to do that and show all the code needed to create the reusable lambda library element.

a) We identify the parameters needed for the pipeline.

b) We identify a Functional Interface for our lambda

c) We create the Functional Interface

d) Move the lambda stream code to the library lambda and use the parameters.

public class USLambdaLibrary {

/\*\*

\* Accepts list of UserStories, a priority Integer, and a implemented Boolean

\*

\* Returns a list of UserStories matching the priority and not implemented

\*/

public static final TriFunction<List<UserStory>, Integer, Boolean, List<UserStory>> TOP\_USERSTORIES = (list, priority, implemented) -> list.stream()

.filter(u ->(u.getPriority() == priority))

.filter(u ->(u.isImplemented()== implemented))

.collect(Collectors.toList());

}

[@FunctionalInterface](mailto:@FunctionalInterface)

**public interface** TriFunction<S,T,U,R> {

R apply(S s, T t, U u);

}

4) (2pts) Give an example of how a client main() will call your reusable lambda library element from problem 3.

List<UserStory> lambdaTopUS = USLambdaLibrary.TOP\_USERSTORIES.apply(allUserStories, 1, false);

5) In your client main() use the forEach() method to print out the following for each UserStory in leastDevEffort from problem 2. You will print the UserStory name, the developer name, and the development effort. List all additional methods you need in UserStory to make your solution work.

In UserStory add getName(), getDeveloper(), and getDevEffort().

1. (3pts) Do one solution using a simple (non-reusable) lambda expression

leastDevEffort.forEach(u->System.out.println(u. getName() + " , " + u.getDeveloper() + " ," +

u.getDevEffort());

b) (3pts) Do a second solution making your lambda expression into a reusable method and show how that method is used by forEach().

Consumer<UserStory> printEffortInfo = u->System.out.println(u.getName() + " , " +

u.getDeveloper() + " ," + u.getDevEffort());

leastDevEffort.forEach(printEffortInfo);

6) (6pts) Write your own generic class with multiple generic parameters and multiple generic methods.

Give an example of a client main() using your generic class with a constructor call and then calling one of your generic methods.

public class SimplePair<K,V> {

private K key;

private V value;

public SimplePair(K key, V value) {

this.key = key;

this.value = value;

}

public K getKey() { return key; }

public V getValue() { return value; }

}

public static void main(String[] args) {

SimplePair<Integer, String> p1 = new SimplePair<>(1, "apple");

int k1 = p1.getKey();

7) (2pts) Consider the following code:

public interface AInterface {

default void dmethod(){ System.out.println("calling dmethod");}

}

public interface BInterface {

void dmethod();

}

public class ABclass implements AInterface, BInterface{

//implementation code here

}

Explain what options we have for overriding or not overriding dmethod() for ABclass and what version of dmethod() will be run for each option.

ABclass must override dmethod().

**More Final Exam Points**

1. Comparator

consistent with equals

comparing, thenComparing methods

2. translating lambda to inner class, and conversely

3. given a lambda, find a suitable type, turn it into a method reference, indicate the type of method reference (one of 4 possibilities)

3.5 translate a method reference into a lambda

4. using default and static methods in an interface, and the syntax rules for handling clashes

5. identify parameters and free variables in a lambda expression

6. using the new forEach method on Iterable

7. be familiar with these functional interfaces:

Consumer

BiConsumer

Supplier

Function

BiFunction

TriFunction

Predicate

BiPredicate

Comparator

8. create lambda/stream pipeline to solve problems; transform into a lambda library expression

9. know how to use the following stream operations

filter

map

flatMap

count

collect(Collectors.toList())

collect(Collectors.joining());

collect(Collectors.joining(", "));

collect(Collectors.summarizingInt(...)) //for IntSummaryStatistics

of

generate

iterate

limit

skip

concat

distinct

sorted

max(Comparator)

findFirst

findAny

reduce

10. IntStream operations

not available in Stream:

range

rangeClosed

Stream -> IntStream: mapToInt

IntStream -> Stream: boxed

toArray

sum, average, max, min

10 use the Optional class when necessary

11. techniques for stream "reuse"

12. techniques for testing a lambda/stream pipeline

13. handling exceptions that are thrown in the middle of a lambda/stream pipeline

14. be able to use the get and put principle to guide proper usage of ?, ?extends, and ?super wildcard types.

15. create a generic program using generics

16. restrictions on usage of type variables and parametrized types

1) A subtype of Enum can be defined so that it is a subtype of java.lang.comparable (**false**)

2) Assume employee is an existing class with proper attribute and setter/getter methods. The following Employeeinfo class will be produce a compiler error

Public class EmployeeInfo

Public void sort(List<Employee> emps, final Employee e){

Collections.sort(emps,(e1,e2) -> e.setSalary(10000);

Return e1.getName().compareToIgnoreCase(e2.getName());

});

}

}

*If give compiler error if the parameter ‘e’ reassigned but here we do not reassign it. (if we see employee is final class here and we are setting some value to it)*

3) This is correct lambda express : ()) -> “Tricky Exam” Supplier<String>  **true**

4) This is correct lambda express ()-> return “Tricky Exam” **false** (refer option if we have one statement)

5) The following lambda express is correction: **false**

Object o = ()-> System.out.println(“Tricking Exam”)

*Because lambda can be name in term of functional interface*

6) This lambda below contains no free variables. **false**

(String str) -> System.out.println(str);

7) SmartAdder is a functional interface : **false**

8) in Java8, distinct() method is an intermediate and stateless operation in the Stream API. **false**

*Distinct is state full*

9) Java Generic provide stronger type checking at runtime. **false**

*At compile time and this is the power of generic*

10) the following statement can compile without errors: **true**

List<? extends Integer> intList = new ArrayList<>();

List<? extends Number> numList = intList;

1. immutable objects are automatically thread-safe so you don’t need any synchronization. **true**

2. you can declare a generic method in a non-generic class but you cannot declare a generic interface without declaring a generic method in it. **False**

3. since object is a super type of all reference type in Java, so you can write code like this: **false**

Util<String> us = new Util<>();

Util<Object> uo = us;

4. The following code will compile without problems since Java only support single inheritance for class and object in the super-type of all classes even though we omit “extend object” most of the time. **false**

public enum Size extends Object {

SMALL,MEDIUM,LARGE}

5. In Java 8 Streams’ limit() is a terminal operation because it will return a stream of limited size **false**

6. if two object have the same hashcode, the equals must return true. **False**

7. the following statements can compile without errors. CheckingAccount is a subclass of Account. **true**

List<? extends CheckingAccount> checkingAccount = new ArrayList<>();

List<? extends Account> accounts = checkingAccounts;

8.The following lambda express is correction **false**

Object o = () -> System.out.println(“Tricky final Exam!”); };

Part 2

11. Name two difference between List<?> and List<Object>

**Answer**: a) List<?> is a list of unknown type. List<Object> is Object type list.

b) List<?> is a supertype of all generic list, but List<Object> is not.

c) List<?> can’t be used to add other objects except null and get list<Object> can do.

One particular danger of raw types (e.g. List without the <?>) is that raw types **disable checking even outside of their own declarations**

**List<Object> is bounded does not need a helper method for capturing.**

**List<?> can only be capture by the introduction of helper method & is the unbounded type.**

The more obvious of the two is List<Object> which is a list of arbitrary objects (as every class has Object as super class). The difference to List now is that List is untyped and therefore no type checks at all are performed, which ultimately leads to certain warnings and can lead to weird runtime behavior.

12. How to make class immutable?

**Answer**:

a) Make all attributes private and final

b) make the class final

c) no setter, only have getters

d) getters do not return mutable object.

13) When we use instance-of-strategy to override equals(). If a subclass is introduced, subclass inherits the equals method if subclass override equals, then an asymmetric equals is created. In this case, how do you solve the problem ? (list two)

Answer: 1) make the class final which present inheritance

2) Allow overridden equals method in each subclass

3) it is always preferable to override hashcode with equals.

* declare the superclass *final (*to prevent subclassing)or
* require that every subclass relies on the superclass version of equals() (and does *not* override equals() separately
* another alternative is use composition instead of inheritance.

14) List at least 4 feature characteristics of functional style of programming.

**Answer**:

1) in functional style programming a declarative approached is followed rather than imperative to solve the problem. This leaves a fact that functional programing is concern with “What” rather than “how” to solve the problem.

2) Functional programming provide the same result[output] no matter how many times or when it is called as long as input is the same.

3) can be treated as first-class citizen where just like objects can be used as return types or arguments.

4) functional programming operates without effecting the environment it is being used.

15) when you assign appropriate type, you have to specify type argument, which means change type variable T to specific type such as Integer, Employee etc.

a. Assign the appropriate type to the method reference below and convert it to lambda expression.

String:compareTo Object:instanceMethod

Comparator<List<Employee>, Integer>

result =(Employee e1,Employee e2) - > e1.getName().equals(e2.getName());

b. Assign the appropriate type to method reference below and convert it to method reference.

() -> new HashSet<Person>(); Constructor Reference

Supplier<Person> result = ()->HashSet::new;

16. Create a generic programming solution to the problem of finding second smallest element in an Array. In other words, devise a public static method secondSmallest so that it can handle the biggest possible range of type (such as Integer, Double, Number, LocalDate, etc.) you only need to write the generic method, no need to create test case.

a. 1,2,3,4,5 result =2

b. 1,1,1,2,2,3,4,4,5 = result =2

c. LocalDate: 1900-1-1, 1950-1-1, 1975-1-1, 2000-1-1 result : 1950-1-1

Answer: public static<T extends Comparable<? extends T>> T secondSmallest(T[] t){

If (t==null) return null

T firstSmallestV = t[0];

T secondSmallestV=t[1];

T temp=0;

For (int i=0; i<t.length();i++){

If (t[i].compareTo(firstSmallestV)<0 )[

Temp=firstSmallestV;

firstSmallestV=t[i];

secondSmallestV=temp;

}

}

Return secondSmallestV;

}

Give the lambda express below:

public class CheckoutRecord {

//This lambda expression : (instance) - > this.equals(instance);

//This lambda expression: exists in CheckoutRecord class

}

Provide an anonymous inner class that behaves the same way as the lambda. Hint find the functional interface first.

Answer: Predicate<T> result = new Predicate<T> (

new MyClass {

(instance) -> this.equals(instance);

});

Create a stream pipeline that, when run find all CheckoutRecordEntry for which due date is 2016-2-3 and sort them by checkoutDate.

List<checkOutRecord> checkoutRecords = allEnteries.Stream()

.filter((enteries) -> enteries.getDueDate.equals(LocalDate.of(2016,2,3))

.sorted(campartor.comparing(checkoutEntry::getCheckoutDate).Collect(collectors.toList());

3) Create a stream pipeline that, when run return a String of all books’s title whose genre is ROMANCE, sorted alphabetically. The result’s format looks like Alien, Jimmy’s First Day of School, Jimmy’ First Day of School, Linnea Sincalir.

**Answer**

String bookTitles = allBooks.Stream.Filter(book ->book.getGener==Genre.ROMANCE)

.sorted(Camprator.comparing(Book::getTitle))

//.map(book::getDescription) // this was missing before

.collect(Collectors.joining(“,”));

4) Create a stream pipeline that, when run, finds a book in CheckoutRecordEntry, list with the cheapest price (you MUST use reduce method)

**Answer:**

Book cheapBook = allEntries.Stream().map((enteries) -> entries.getBook())

.reduce(Book b1, (Book b1,Book b2) -> min(b1.getPrice(),b2.getPrice())

.collect(Collectors.joining(“ “)); **// missing something here**

**Optional<Book> cheapBook = allEntries.Stream().map(CheckoutRecordEntry::getBook)**

**.reduce((t1,t2) -> t1.getPrice() < t2.getPrice() ? t1 : t2);**

**System.out.println(smallestTransaction.get());**

5) Turn the stream pipeline of Question 3) into a library element of a Lambda Library which can be used to search based on other CheckoutRecordEntry lists and different due date filter criteria Make sure you choose the correct functional interface.

**Answer**

BiFunction<List<CheckoutRecordEntry>, Genre, String> Filter\_By\_Genre =

(list,genre) - > list.stream()

.filter(entries -> entries.getBook().getGenre()==genre)

.map(entries -> enteries.getBook().getTitle())

.sorted()

.collect(collectors.joining(“ ,”));

String result=LambdaLibary.Filter\_By\_Genre.apply(allEntries,DRAMA);

11. Java 8 introduces default method and static methods for interfaces, Explain how these new feature work. For each of these feature, describe one significant advantage that the feature provides.

**Answer**:

Default method is used for implementation for the problem of evolving API, when the customer gets new interfaces and has no implementation, the client code was failing. This new default method provide implementation to the interface to solve any future problem of application. We can add new methods to interface.

Static Method; we can add static method to interface, this method is same as class static method, except is not inherit the same feature, we do not need utility classes and we can use this as interface.

12) Why is it import to override hashCode(), whenever you override the equals() method? If you fail to do so, give an example of a difficulty that could arise.

**Answer**:

hashCode() is the identity of the object used to add object. HashMap, if we override equals, but no hashcode, then two with same attribute (data) will be added. You must override hashCode() in every class that overrides equals(). Failure to do so will result in a violation of the general contract for Object.hashCode(), which will prevent your class from functioning properly in conjunction with all hash-based collections, including HashMap, HashSet, and Hashtable.

Rules to a key in has table.

1. Keys: override equal & hash code
2. Equals and hashcode overridden in the same way (i.e. using same attributes)
3. The class must be immutable.

13) What functional interfaces would you use for the following description what are equivalent method references for the following lambda expressions? (you must specify **type argument** for functional interfaces like homework)

a. (list element) -> list.contain(element);

b. (name,salary) -> new Dish(name,salary) ; (see Dish class in question 15)

c. (string name) -> name.length();

d. s-> Integer.parseInt(s);

e. (Object d) -> this.equals(d);

14) Write a function evenNumbers(int n) that outputs the first n even whole numbers, represented as BigInteger. For instance, the call evenNumber(5) produce 0,2,4,6,8. Your function must be implemented using a stream.

public void evennumbers(int n){

Stream.iterate(BigInteger.ZERO,

num-> num.add(BigInteger.ONE).add(BigInteger.ONE))

.limit(n)

.forEach(System.out::println);

15) Write a generic method to find the maximal element in the range

begin…. End

of a list, where begin and end are positions in the input array.

For example:

Input = (50,8,4,9,11,25,1,19,7,29)

Begin =2

End =6

Max=25 (values in the range 2..6 are 4,9,11,25)

**static** <T **extends** Comparable<? **super** T>> T findmax(T[] input, **int** begin, **int** end)

{

**if** (input==**null**) **return** **null**;

**if** (input.length==0) **return** **null**;

T max = input[begin];

**for** (**int** i=begin; i<=end; i++) {

**if** (input[i] **instanceof** Integer)

{

**if** (input[i].compareTo(max)==-1)

max=input[i];

}

**if** (input[i] **instanceof** String)

{

**if** (input[i].compareTo(max)>0)

max=input[i];

}

**else**

{

**if** (input[i].compareTo(max)>0)

max=input[i];

}

}

**return** max;

}

16) You are asked to write a Dish inventory for a restaurant. The domain class is listed below. Assume proper constructors and setter/ getters are provided.

public class Dish {

private String name;

private Boolean vegetarian;

private int calories;

private Type type;

public Dish() {}

public Dish(String name, int calories){

this.name=name;

this.calories =calories;

}

// setter /getter method not shown

Public String toString() {

Return “[“ +name+”,”+ calories+ “,” + type.toString()+”]”;

}

}

public enum Type {

MEAT,FISH,OTHER

}

public class Main {

public static final List<Dish> menu =

Arrays.asList(new Dish(“portk”,false,800, Type.MEAT),

new Dish(“beef”,false,700,Type.MEAT),

new Dish(“chicken”,false,400,Type.MEAT),

new Dish(“French fries”,true,530,Type.OTHER), new Dish(“rice”,true,350,Type.OTHER),

new Dish(“season fruit”, true,120,Type.OTHER),

new Dish(“pizza”,true,550,Type.OTHER),

new Dish(“prawns”,false, 440,Type.FISH), new Dish(“salmon”,false,450,Type.FISH);

}

Use lambdas and streams to solve the following

(you have to assign the output to a variable to hold the result. E.g. String s=”final exam”)

1. Output to a List of all the dishes on the menu which are not vegetarian and contain more than 400 calories. The result should be a List of String.

result = menu.Stream().filter(d-> d.getVegetrain==false)

.filter(d->d.getCalories()>400)

.collect(collectors.toList());

1. Output to a List the first 3 dishes of type MEAT in sorted order, sorted by number of calories.

List<Dish> dishes = menu.Stream().filter(……)

.sorted(campare.comparing)

1. Return a comma-separated string of all dishes names.

String result = menu.Stream()

.map(d->d.getName())

.collect(collectors.Joining(“ ,”));

1. Write an expression that will determine whether any dish is named “French fries” output should be “true” or “false”

Int count = menu.Stream().

filter(d – “French Fries”.equals(d.getName()).count();

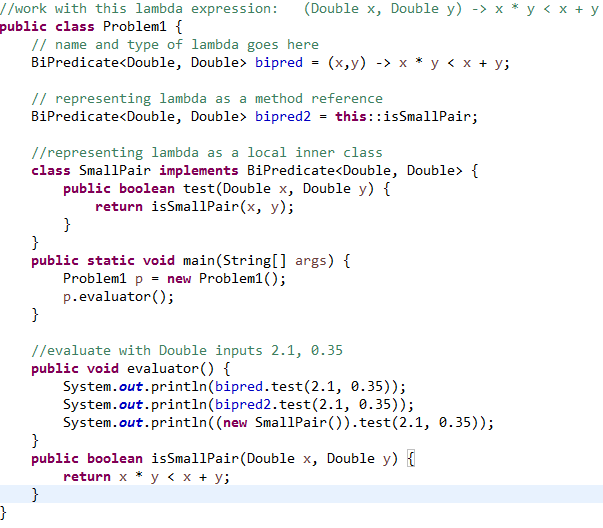
1. Print the largest number of calories of Dish in the menu. (For this problem, you must use the reduce method of Streams)
2. Turn the stream pipeline of Question 1 into a Library element which can be used to search based on any menu list and filter criteria (calories > 400,”400” should be passed as argument).

Part 3 Write one or two paragraph relating a point from course to a principal of SCI

Answer: All classes in Java inherited from object class likewise all individual intelligence inherited from unified field.

The purification process of using Lambda…

**Answers**



.