1. Интерфейс Serializable. Объектный граф, сериализация и десериализация полей и методов.

Сериализовать можно только объекты тех классов, которые реализуют интерфейс java.io.Serializable.

Serializable - это интерфейс маркера (не имеет элемента данных и метода). Он используется для “пометки” классов java, чтобы объекты этих классов могли получить определенные возможности.

Если родительский класс реализовал сериализуемый интерфейс, то дочернему классу не нужно его реализовывать, но наоборот неверно.

В процессе сериализации сохраняются только нестатические элементы данных.

Граф объектов - это набор объектов, которые будут сериализованы автоматически, если объект, содержащий ссылку на них, сериализуется.

Другими словами, мы можем сказать, что когда мы сериализуем любой объект и если он содержит какую-либо другую ссылку на объект, то JVM сериализует объект, а также ссылки на его объекты.

В графе объектов каждый объект должен быть сериализуемым. Если хотя бы один объект не является сериализуемым, то мы получим выполнение во время выполнения с надписью NonSerializableExecution

|  |  |  |
| --- | --- | --- |
| type | S | D |
| public | + | + |
| getter | + | + |
| Setter |  | + |
| **private** |  |  |
| deafult |  |  |
| **protected** |  |  |

2.**Java Stream API. Создание конвейеров. Промежуточные и терминальные операции.**

Stream API используется для обработки коллекций объектов. Поток - это последовательность объектов, которая поддерживает различные методы, которые могут быть переданы по конвейеру для получения желаемого результата.

Stream - это не структура данных, вместо этого он принимает входные данные из Коллекций, массивов или каналов ввода-вывода.

Streams не изменяют исходную структуру данных, они только предоставляют результат в соответствии с конвейерными методами

Каждая промежуточная операция выполняется и в результате возвращает поток, следовательно, различные промежуточные операции могут быть переданы по конвейеру. Терминальные операции отмечают конец потока и возвращают результат.

**Промежуточные Операции**

1.Map Метод map используется для возврата потока, состоящего из результатов применения данной функции к элементам этого потока

ist square = number.stream().map(x>x\*x).collect(Collectors.toList());

2.Filter :Метод filter используется для выбора элементов в соответствии с предикатом, переданным в качестве аргумента.

List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toset());

3.sorted

**Терминальные операции**

Collect: Метод collect используется для возврата результата промежуточных операций, выполненных в потоке.

forEach: Метод forEach используется для перебора каждого элемента потока

number.stream().map(x->x\*x).forEach(y->System.out.println(y));

reduce: Метод reduce используется для приведения элементов потока к единому значению.

Метод reduce принимает двоичный оператор в качестве параметра.

 Optional<String> longestString = words.stream().reduce((word1, word2)-> word1.length() > word2.length()? word1 : word2)

**Шаблоны проектирования**

Шаблон проектирования обеспечивает общее многоразовое решение для общих проблем, возникающих при разработке программного обеспечения. Шаблон обычно показывает отношения и взаимодействия между классами или объектами. Идея состоит в том, чтобы ускорить процесс разработки, предоставляя хорошо проверенные, проверенные парадигмы разработки / проектирования.шаблон проектирования представляет собой идею, а не конкретную реализацию. Используя шаблоны проектирования, вы можете сделать свой код более гибким, многоразовым и ремонтопригодным.

**Creational** : *Factory Method, Singleton*

**Structural** : *Adapter, Proxy, Decorator, Façade, Flyweight*

**Behavioral:** *Command , Interpreter, Iterator, Strategy*

*Factory Method* :сделаем класс который создат все обекты

*Strategy: when we have a v behavior that varies for examble animals part of them flies and part not e make interface fly and we make to classis that implimemnt fly one it flues with fly method that says am flying and one with fly method that tells that animal trytofly this method go to the field it flies and execute it …..*

*Singleton: when want only one cinstance/object so we make a class with empty constructor and with method that returns a field called instance of type our class this method does not create object unless the field is null thus creates only one onj*

*Decorator:we have a pizza shop that have topping every topping affects the desc and price so we make*

*class for plainpizza implements interface pizza*

*class toppingdicrator implement pizza that gets pizza and execute its method*

*class toping impiment topping dicrator and add to it the price and decription*

*and then we make a pizza object with new plain pizza and then we add the topping as objects new new*

*Adapter: when you have class that have different behavior but should be part of inhiretnece we make the class and we make adapter that inharrets father and point at the override the method by pointing at the methods in class we need*

*Façade:when you make simplified interface that perform man actions*

*We make many classes that checks it self and perform actions and then we make class that uses this clases as fields*

*Iterator: when we need to handel array and arraylist hashtaple blab la, we make them all impiment interface iterator that have only one method create itereator and then when we get the object we use this method to iterate*

*Flyweight: when we try to se the common things in our object and change them instead of making new object every time for example if you have a red tringale and you read to make 500 hundred with different sizes we can use the same object and change the size only in order not to create new object and make the code faster*

*Proxy: when you have a class that you want to secure that have method that you do not want to use in main like setters you make proxy class that implement the same interface as normal class and have only the methods that you need and use it to create object*

*Interpreter: let us say we have class that needs to impiment or and and logical operation on string we makeinterface called expprision that have method interpret and then we make or expprison and and and make them implement expprison and overrided intterbut*