**Билет №1**

**Парадигма ООП. Основные принципы ООП и их реализация в языке программирования Java и С++.**

ООП - парадигма разработки ПО, в которой программа представляется в виде совокупности объектов.

Основные принципы:

*Инкапсуляция* – базовый принцип ООП, при котором атрибуты и поведение объекта объединяются в одном классе, внутренняя реализация объекта скрывается от пользователя, а для работы с объектом предоставляется открытый интерфейс.

*Полиморфизм* – механизм, позволяющий коду работать с разными типами данных. Бывает параметрический полиморфизм, когда один и тот же код используется для разных допустимых типов аргументов и ad-hoc полиморфизм, подразумевающий выполнение разного кода для разных типов данных, пусть и вызываемого единообразно.

*Наследование* – принцип, позволяющий объектам наследовать данные и функциональность у уже существующих типов.

Эти принципы реализуются **в С++** следующим образом:

* Объекты задаются ключевым словом class, structure;
* Допустимо множественное наследование;
* Есть абстрактные классы и виртуальные функции, подразумевающие наследование и переопределение;
* Ключевые слова private, protected и public управляют доступом.

**В Java**:

* Объекты задаются ключевым словом class и interface;
* Ключевые слова private, protected, publiс, default управляют доступом;
* Нет множественного наследования, но класс может реализовывать сколько угодно интерфейсов Абстрактные классы и виртуальные функции также есть;
* Есть интерфейсы – абстрактные типы данных, указывающие поведение, которое необходимо реализовать классам.

**Понятие структуры данных список. Линейный список. Виды списков и их реализация на Java. Доступ к элементу структуры данных список. Использование списков. Трудоемкость операций со списками.**

Список – это динамическая линейная структура данных, в которой каждый элемент ссылается либо только на предыдущий – однонаправленный линейный список, либо на предыдущий и следующий за ним – двунаправленный линейный список.

ArrayList — реализация изменяемого массива интерфейса List, часть Collection Framework, который отвечает за список (или динамический массив), расположенный в пакете java.utils. В основе ArrayList лежит идея динамического массива. А именно, возможность добавлять и удалять элементы, при этом будет увеличиваться или уменьшаться по мере необходимости. ArrayList хранит только ссылочные типы, любые объекты, включая сторонние классы. Строки, потоки вывода, другие коллекции. Для хранения примитивных типов данных используются классы-обертки. ArrayList следует использовать, когда в приоритете доступ по индексу, так как эти операции выполняются за константное время. Добавление в конец списка в среднем тоже выполняется за константное время. Кроме того, в ArrayList нет дополнительных расходов на хранение связки между элементами. Минусы в скорости вставки/удаления элементов, находящихся не в конце списка, так как при этой операции все элементы правее добавляемого/удаляемого сдвигаются.

**Билет 2**

**Получение информации о типе. Создание экземпляров классов. Вызов методов класса.**

public class Main {  
 public static void main(String[] args) {  
 MyClass myClass = new MyClass();  
 System.*out*.println(myClass.getClass().getName());  
 myClass.print();  
 }  
}  
public class MyClass {  
 int a = 8;  
 public void print() {  
 System.*out*.println(a);  
 }  
}

А если тебе нужно в условии проверить допустим является ли переменная строкой, то пригодится следующее: if (s instanceof String)

вместо String, можешь поставить любой класс, который тебе нужен.

**Организация программы на Java. Основные структурные единицы. Процесс интерпретации и компиляции. Роль JVM.**

Java — мультиплатформенный язык программирования. Это значит, что программы, написанные на языке Java, можно выполнять на любой платформе, где установлена специальная исполняющая система Java. Такая система называется Java Virtual Machine (JVM). Для того, чтобы перевести программу из исходного кода в код, понятный JVM, нужно её скомпилировать. Код, понятный JVM называется байт-кодом и содержит набор инструкций, которые в дальнейшем будет исполнять виртуальная машина.  
Для компиляции исходного кода в байт-код существует компилятор javac, входящий в поставку JDK (Java Development Kit). На вход компилятор принимает файл с расширением .java, содержащий исходный код программы, а на выходе выдает файл с расширением .class, содержащий байт-код, необходимый для исполнения программы виртуальной машиной.  
После того, как программа была скомпилирована в байт-код, она может быть выполнена с помощью виртуальной машины.  
Компиляция — трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду.  
Интерпретация — пооператорный (покомандный, построчный) анализ, обработка и тут же выполнение исходной программы или запроса (в отличие от компиляции, при которой программа транслируется без её выполнения).  
Язык Java обладает как компилятором (javac), так и интерпретатором, в роли которого выступает виртуальная машина, которая построчно преобразует байт-код в машинный код и тут же его исполняет. Таким образом, когда мы запускаем скомпилированную программу, виртуальная машина начинает её интерпретацию, то есть построчное преобразование байт-кода в машинный код, а также его исполнение.  
К сожалению, чистая интерпретация байт-кода является довольно долгим процессом и делает язык java медленным в сравнении с его конкурентами. Дабы избежать этого, был введен механизм, позволяющий ускорить интерпретацию байт-кода виртуальной машиной. Этот механизм называется Just-in-time компиляцией (JITC).  
1) Компилятор javac преобразует исходный код программы в байт-код, который может быть выполнен на любой платформе, на которой установлена виртуальная машина Java;  
2) После компиляции JVM интерпретирует получившийся байт-код;  
3) Для ускорения работы Java-приложений, JVM использует механизм Just-In-Time компиляции, который преобразует наиболее часто выполняемые участки программы в машинный код и хранит их в памяти.

**Билет 3**

**Структурирование Java приложения, пакеты. Уровни доступа видимости.**

В Java существует 4 уровня доступа к полям и методам класса:

* public - доступно всем
* protected - доступно внутри пакета и в подклассах
* default - доступно внутри пакета
* private - доступно только внутри класса

**Понятие сортировки массивов. Сортировка пузырьком. Сортировка вставками. Использование ООП для программирования алгоритмов сортировок в массивах и коллекциях.**

Чтобы просмотреть элементы массива, вы можете использовать цикл для перебора элементов массива. Например:

int[] numbers = {1, 2, 3, 4, 5};

for (int i = 0; i < numbers.length; i++) {

int element = numbers[i];

System.out.println(element);

}

Для поиска определенного элемента в массиве вы можете использовать цикл и условный оператор. Например:

int[] numbers = {1, 2, 3, 4, 5};

int target = 3;

for (int i = 0; i < numbers.length; i++) {

if (numbers[i] == target) {

System.out.println("Found at index " + i);

break;

}

}

Для сортировки массива можно использовать один из алгоритмов сортировки, предоставляемых стандартной библиотекой Java, например Arrays.sort(). Например:

int[] numbers = {4, 3, 2, 1, 5};

Arrays.sort(numbers);

Чтобы вычислить сумму элементов массива, вы можете использовать цикл для перебора элементов массива и добавления их к промежуточной сумме. Например:

int[] numbers = {1, 2, 3, 4, 5};

int sum = 0;

for (int i = 0; i < numbers.length; i++) {

sum += numbers[i];

}

System.out.println("Sum: " + sum);

**Билет 4**

**Понятие класса. Определение, инициализация. Модификаторы доступа. Константы и переменные. Объявление классов.**

Понятие класса Класс в Java — это шаблон для создания объекта, а объект — это экземпляр класса. Класс определяет структуру и поведение, которые будут совместно использоваться набором объектов. Класс содержит переменные и методы, которые называются элементами класса, членами класса. Он составляет основу инкапсуляции в Java. Каждый объект данного класса содержит структуру и поведение, которые определены классом. Иногда объекты называют экземплярами класса.

Методы используются для описания того, что объект класса умеет делать или что можно с ним сделать. Переменные - для описания свойств или характеристик объекта.

**Объявление класса.** После ключевого слова class пишется имя класса. В теле класса объявляются переменные и методы класса. Их может быть сколько угодно.

class Имя\_Класса{

тип переменнаяЭкземпляра1;   
тип переменнаяЭкземпляра2;   
// ...   
тип переменнаяЭкземпляраN;   
тип имяМетода 1 (список параметров) {// тело метода}   
тип имяМетода2 (список параметров) {// тело метода}   
…   
тип имяМетодаN (список параметров) {// тело метода}}

**Инициализация объекта.** Инстанцирование (англ. instantiation) — создание экземпляра класса. В отличие от слова «создание», применяется не к объекту, а к классу. То есть, говорят: «(в виртуальной среде) создать экземпляр класса или инстанцировать класс». Порождающие шаблоны используют полиморфное инстанцирование.

Конструкция: ClassName ObjectName = new classConstructor();

Модификаторы доступа:

Private (закрытый) – есть свой класс

Protected (защищенный) – есть свой класс, пакет, класс наследник

Public (открытый) – всё доступно

**Константы и переменные.** Для хранения данных в программе предназначены переменные. Переменная представляет именованную область памяти, которая хранит значение определенного типа. Каждая переменная имеет тип, имя и значение. Тип определяет, какую информацию может хранить переменная или диапазон допустимых значений.

Также можно присвоить значение переменной при ее объявлении. Этот процесс называется инициализацией.

Кроме переменных, в Java для хранения данных можно использовать константы. В отличие от переменных константам можно присвоить значение только один раз. Константа объявляется также, как и переменная, только вначале идет ключевое слово final.

**Понятие поиска в массивах. Последовательный поиск. Сортировка методом прямого выбора. Использование ООП для программирования алгоритмов поиска в массивах и коллекциях.**

Поиск является видом обработки данных в массиве с целью определения и выявления элемента, который соответствует заданным критериям.

Линейный или последовательный поиск – простейший алгоритм поиска. Он редко используется из-за своей неэффективности. По сути, это метод полного перебора, и он уступает другим алгоритмам.

У линейного поиска нет предварительных условий к состоянию структуры данных. Алгоритм ищет элемент в заданной структуре данных, пока не достигнет конца структуры. При нахождении элемента возвращается его позиция в структуре данных. Если элемент не найден, возвращаем -1.

Прямой выбор основан на следующих правилах:

* Выбирается элемент с наименьшим ключом.
* Он меняется местами с первым элементом a1
* Затем этот процесс повторяется с оставшимися n-1 элементами, n-2 элементами и т.д. до тех пор, пока не не останется один, самый большой элемент.

**Билет 6**

**Оператор new. Понятие ссылки и указателя на объект. Реализация в С++ и Java. Время жизни объекта.**

Ключевое слово new отвечает за выделение памяти. Используется для создания объектов класса, а также массивов.

Ссылки в Java — указатели на объекты. Другими словами, ссылка — это переменная, содержащая адрес ячейки памяти, в которой хранится объект. Кроме того, ссылка может быть инициализирована как null — нулевая ссылка, не указывающая ни на какой объект в памяти (именно это значение является значением по умолчанию). Внутри класса в нестатическом контексте также может быть использована ссылка this, указывающая на текущий объект, и ссылка super, указывающая на текущий объект суперкласса.

В Java различают переменные примитивного и ссылочного типа. К примитивным относят 8 типов — пять целочисленных, два для чисел с плавающей точкой и логический тип boolean. Все остальные типы данных относятся к ссылочным типам. Это значит, что объявленная переменная этого типа хранит не состояние объекта, а ссылку на него, сам же объект хранится в определённой области памяти (куче). Создание нового класса, абстрактного класса, интерфейса или перечисления равносильно определению нового ссылочного типа данных. В отличие от создания объектов, объявление ссылок типа абстрактный класс либо интерфейс не запрещено.

Началом времени существования любого объекта является момент его создания (отведение участка памяти), а окончанием – возвращение отведенного участка памяти системе.

Student student = new Student(); или new Student(); или Student student; student = new Student();

В первом случае мы работаем с переменной ссылочного типа данных, которая содержит в себе ссылку на объект класса, который мы инициализировали, во втором случае - мы не создаём отдельной переменной - ссылки на объект класса, однако объект всё ещё находиться в системе. А в третьем объявили ссылочную переменную, и потом уже присвоили ей ссылку на инициализированный объект.

**Организация работы с файлами в Java.**

Класс File, определенный в пакете java.io, не работает напрямую с потоками. Его задачей является управление информацией о файлах и каталогах.

В зависимости от того, что должен представлять объект File - файл или каталог, мы можем использовать один из конструкторов для создания объекта:

File (String путь\_к\_каталогу)

File (String путь\_к\_каталогу, String имя\_файла)

File (File каталог, String имя\_файла)

Например:

// создаем объект File для каталога

File dir1 = new File ("C://SomeDir");

// создаем объекты для файлов, которые находятся в каталоге

File file1 = new File ("C://SomeDir", "Hello.txt");

File file2 = new File (dir1, "Hello2.txt");

Для того чтобы прочитать данные из файла или записать их в него, необходимо работать с потоками InputStream, OutputStream:

File file = new File ("/home/user/pic.jpg");

InputStream fIn = new FileInputStream(file);

Часто переменную определяют самим классом-адаптером:

FileInputStream fIn = new FileInputStream(file);

У класса очень много методов, перечислим некоторые:

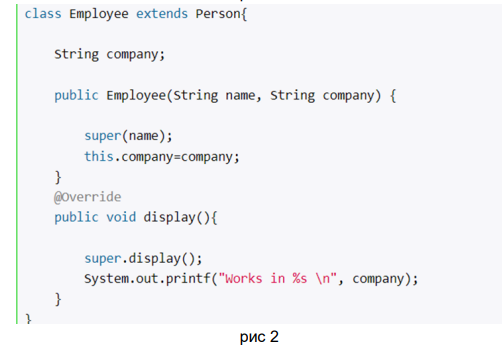
**getAbsolutePath()** - абсолютный путь файла, начиная с корня системы. В Android корневым элементом является символ слеша (/)

**canRead()** - доступно для чтения; **canWrite()** - доступно для записи**; exists()** - файл существует или нет; **getName()** - возвращает имя файла; **getParent()** - возвращает имя родительского каталога; **getPath()** – путь; **lastModified()** - дата последнего изменения; **isFile()** - объект является файлом, а не каталогом; **isDirectory()** - объект является каталогом; **isAbsolute()** - возвращает true, если файл имеет абсолютный путь; **renameTo(File newPath)** - переименовывает файл. В параметре указывается имя нового имени файла. Если переименование прошло неудачно, то возвращается false; **delete()** - удаляет файл. Также можно удалить пустой каталог

**Билет 7**

**Переопределение методов в Java, абстрактные методы.**

Производный класс может определять свои методы, а может переопределять методы, которые унаследованы от базового класса (рис.1).



Перед переопределяемым методом указывается аннотация @Override. Данная аннотация в принципе не обязательна. При переопределении метода он должен иметь уровень доступа не меньше, чем уровень доступа в базовом класса. Например, если в базовом классе метод имеет модификатор public, то и в производном классе метод должен иметь модификатор public. Однако в данном случае мы видим, что часть метода display в Employee повторяет действия из метода display базового класса. Поэтому мы можем сократить класс Employee (рис.2)

Абстрактные методы – это методы, у которых отсутствует реализация. Кроме обычных методов абстрактный класс может содержать абстрактные методы. Такие методы определяются с помощью ключевого слова abstract и не имеют никакой реализации. Производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе. Также следует учитывать, что если класс имеет хотя бы один абстрактный метод, то данный класс должен быть определен как абстрактный.

**Иерархия классов ввода вывода. Работа с файлами в Java. Работа с файлами. Сериализация объектов.**

В языке Java существует ряд классов, которые обеспечивают работу байтовых потоков ввода/вывода. Все классы байтовых потоков разделены на две иерархии. В вершине одной иерархии (соответствует вводу) размещается класс InputStream. В вершине второй иерархии (соответствует выводу) размещается класс OutputStream.

**Сериализация** представляет процесс записи состояния объекта в поток, соответственно процесс извлечения или восстановления состояния объекта из потока называется десериализацией. Сериализация очень удобна, когда идет работа со сложными объектами.

**Интерфейс Serializable**. Сразу надо сказать, что сериализовать можно только те объекты, которые реализуют интерфейс Serializable. Этот интерфейс не определяет никаких методов, просто он служит указателем системе, что объект, реализующий его, может быть сериализован.

**Билет 9**

**Статические поля и методы. Класс Math, его основные методы.**

Статические поля и методы. Класс Math, его основные методы.

Модификатор static (постоянный) делает переменную или метод независимым от объекта класса.  
1) Методы с модификатором public static можно вызывать без создания объекта класса. К статическому и обычному методу можно обращаться через объекты класса, но через класс (без объекта) можно обращаться только к статическому методу.  
2) Статические методы нельзя переопределить в классах наследниках. Такой метод един и для текущего класса, и для всех его наследников.  
3) Статические методы могут работать только со статическими методами или полями. Так как статические методы работают без создания объектов, а если в этих методах попытаться обратиться к нестатическому полю, то будет ошибка, так как объект не создавался, соответственно и поля тоже не существует. (Но через нестатические методы можно обращаться к статическим методам и полям)

Класс Math располагается в пакете java.lang и предоставляет набор статических методов для осуществления ряда различных математических вычислений. Методы реализованы как static, поэтому можно сразу вызывать через Math.methodName() без создания экземпляра класса.

Ниже приведены примеры вычислений, для которых класс Math может оказаться полезным:

Вычисление абсолютных значений Math.abs(-1)

Вычисление значений тригонометрических функций Math.sin(Math.toRadians(30))

Возведение в различные степени Math.pow(3,2) //9

Извлечение корней различных степеней Math.cbrt(125) //5

Генерация случайных чисел Math.random() - от 0.0 до 1.0  
Округления  
Метод round - округляет так, как привычно обывателю. Если дробная часть числа больше либо равна 0.5, то число будет округлено в большую сторону, иначе в меньшую.  
Метод floor всегда, независимо от значений дробной части, округляет число в меньшую сторону (в сторону отрицательной бесконечности).  
Метод ceil, напротив, вне зависимости от значений дробной части, округляет числа в большую сторону (в сторону положительной бесконечности).  
**Обобщённое програмирование. Понятие и использование джеенериков в Java**  
Обобщенное программирование. Понятие и использование дженериков в Джава.  
В JDK введены так называемые обобщенные или параметризованные типы – generics или по-другому обобщенные типы.  
Использование параметризации позволяет создавать классы, интерфейсы и методы, в которых тип обрабатываемых данных задается как параметр.  
Дженерики или обобщенные типы позволяют вам абстрагироваться от использования конкретных типов.  
Например:  
<E> для элемента коллекции;  
<T> для обобщенного типа;  
<K, V> ключ и значение.  
<N> для чисел  
S, U, V, и т.д. для второго, третьего, четвертого типа параметра  
Обобщённые методы или дженерики позволяют работать с различными типами данных без изменения их описания, одно из назначений этих методов — более сильная проверка типов во время компиляции и устранение необходимости явного приведения. Создаётся более безопасный и легко читаемый код, который не перегружен переменными типа Object и приведением классов.  
Обобщенное программирование - использование дженериков в программах.

**Билет 15**

**ООП в Java. Понятие объекта. Что представляет собой Java приложение с точки зрения ООП. Основные характеристики объектов в Java**

Концепции ООП являются основополагающими элементами и составляют основу языка программирования Java. В рамках данного подхода выделяют следующие термины: абстракция, инкапсуляция, наследование и полиморфизм.

Класс в Java — это шаблон для создания объекта, а объект — это экземпляр класса. Класс определяет структуру и поведение, которые будут совместно использоваться набором объектов. Класс содержит переменные и методы, которые называются элементами класса, членами класса. Он составляет основу инкапсуляции в Java. Каждый объект данного класса содержит структуру и поведение, которые определены классом. Иногда объекты называют экземплярами класса. Методы используются для описания того, что объект класса умеет делать или что можно с ним сделать.

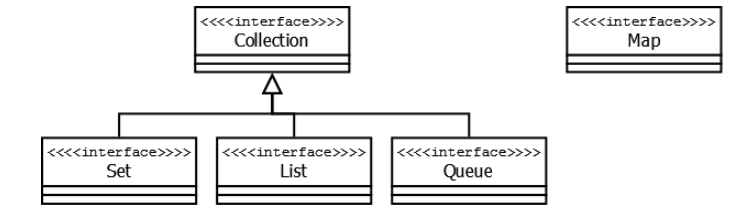
Переменные - для описания свойств или характеристик объекта.

Любой объект может обладать двумя основными характеристиками: состояние - некоторые данные, которые хранит объект, и поведение - действия, которые может совершать объект. Для хранения состояния объекта в классе применяются поля или переменные класса.

**Возможности Java Framework Collection. Контейнер HashMap b его основные методы.**

Java Collection Framework — иерархия интерфейсов и их реализаций, которая является частью JDK и позволяет разработчику пользоваться большим количеством структур данных из «коробки».

На вершине иерархии в Java Collection Framework располагаются 2 интерфейса: Collection и Map. Эти интерфейсы разделяют все коллекции, входящие во фреймворк на две части по типу хранения данных: простые последовательные наборы элементов и наборы пар «ключ — значение» (словари).



Collection — этот интерфейс находится в составе JDK c версии 1.2 и определяет основные методы работы с простыми наборами элементов, которые будут общими для всех его реализаций (например size(), isEmpty(), add(E e) и др.).

HashMap - структура данных в виде словаря, в котором каждый объект имеет уникальный ключ и некоторое значение. В HashMap хеш-таблица реализована на основе динамического массива односвязных списков. По сути, мы получаем хеш-код ключа в результате работы метода hashCode(), который затем модифицируется, а внутри с помощью дополнительного метода полученные значения распределяются по нужным ячейкам. Операции добавления, удаления и поиска будут выполняться за константное время из-за хеширования.

Принцип создания:

HashMap<Integer, String> states = new HashMap<Integer, String> ();

Методы и описание

1 void clear() Удаляет все соответствия с этого Map.

2 Object clone() Возвращает мелкую копию этого экземпляра HashMap: сами ключи и значения не клонируются.

3 boolean containsKey(Object key) Возвращает true, если этот Map содержит отображение для указанного ключа.

4 boolean containsValue(Object value) Возвращает true, если этот Map отображает один или несколько ключей в указанное значение.

5 Set entrySet() Возвращает представление (вид) коллекции отображений, содержащихся в этом Map.

6 Object get(Object key) Возвращает значение, для которого указанный ключ отображается в этой хэшкарте идентификатора, или null (нуль), если Map не содержит отображения для этого ключа.

7 boolean isEmpty() Возвращает true, если этот Map не содержит отображений значений ключа.

8 Set keySet() Возвращает заданное представление (вид) ключей, содержащихся на этом Map.

9 Object put(Object key, Object value) Связывает указанное значение с указанным ключом на этом Map.

10 putAll(Map m) Копирует все отображения с указанного Map на этот Map. Эти отображения заменят любые отображения, которые это отображение имело для любого из ключей, находящихся в настоящее время на указанном Map.

11 Object remove(Object key) Удаляет отображение для этого ключа с этого Map, если присутствует.

12 int size() Возвращает количество ключ-значение отображений на этом Map.

**Билет 16**

**Работа со строками в Java, строковый кэш. Операция конкатенации строк.**

Для соединения строк можно использовать операцию сложения ("+"):

String str1 = "Java";

String str2 = "Hello";

String str3 = str1 + " " + str2;

System.out.println(str3); // Hello Java

При этом если в операции сложения строк используется нестроковый объект, например, число, то этот объект преобразуется к строке:

String str3 = "Год " + 2015;

Фактически же при сложении строк с нестроковыми объектами будет вызываться метод valueOf() класса String. Данный метод имеет множество перегрузок и преобразует практически все типы данных к строке. Для преобразования объектов различных классов метод valueOf вызывает метод toString() этих классов.

Другой способ объединения строк представляет метод concat():

String str1 = "Java";

String str2 = "Hello";

str2 = str2.concat(str1); // HelloJava

Метод concat() принимает строку, с которой надо объединить вызывающую строку, и возвращает соединенную строку.

Еще один метод объединения - метод join() позволяет объединить строки с учетом разделителя. Например, выше две строки сливались в одно слово "HelloJava", но в идеале мы бы хотели, чтобы две подстроки были разделены пробелом. И для этого используем метод join():

String str1 = "Java";

String str2 = "Hello";

String str3 = String.join(" ", str2, str1); // Hello Java

Метод join является статическим. Первым параметром идет разделитель, которым будут разделяться подстроки в общей строке, а все последующие параметры передают через запятую произвольный набор объединяемых подстрок - в данном случае две строки, хотя их может быть и больше

**Возможности Java Framework Collection. Контейнер HashSet его основные методы.**

Java Collection Framework — иерархия интерфейсов и их реализаций, которая является частью JDK и позволяет разработчику пользоваться большим количеством структур данных из «коробки».

Название Hash... происходит от понятия хэш-функция. Хэш-функция — это функция, сужающая множество значений объекта до некоторого подмножества целых чисел. Класс Object имеет метод hashCode(), который используется классом HashSet для эффективного размещения объектов, заносимых в коллекцию. В классах объектов, заносимых в HashSet, этот метод должен быть переопределен (override).

Имеет два основных конструктора (аналогично ArrayList):

// Строит пустое множество

public HashSet()

// Строит множество из элементов коллекции

public HashSet(Collection c)

Методы

public Iterator iterator(); public int size(); public boolean isEmpty(); public boolean contains(Object o); public boolean add(Object o); public boolean addAll(Collection c); public Object[] toArray(); public boolean remove(Object o); public boolean removeAll(Collection c); public boolean retainAll(Collection c) - (retain — сохранить). Выполняет операцию "пересечение множеств".

public void clear() public Object clone()

Методы аналогичны методам ArrayList за исключением того, что метод add(Object o) добавляет объект в множество только в том случае, если его там нет. Возвращаемое методом значение — true, если объект добавлен, и false, если нет.

Перейдём к практике. Как это ни странно, но в жизни встречаются несколько Барсиков, Мурзиков и прочих Рыжиков. Несмотря на одинаковые имена, каждый кот неповторим. Надеюсь, с этим никто не спорит. Но пихать имена котов в множество HashSet не стоит, так как в множестве может храниться только одно имя и двух Мурзиков тут не записать. Другое дело - страны. Не может быть двух Франций, двух Англий, двух Россий (даже партия такая есть Единая Россия, впрочем, мы отвлеклись).

**Билет 18**

**Циклические конструкции в Java. Использование циклов для работы с массивами. Использование циклов для работы с массивами. Использование итераторов для обработки массивов.**

Еще одним видом управляющих конструкций являются циклы. Циклы позволяют в зависимости от определенных условий выполнять определенное действие множество раз.

В Джава есть три классических вида цикла и один улучшенный:

1) Итерационный for()

2) С предусловием while()

3) С постусловием do{} while()

4) for : each для работы с массивами и коллекциями

1) Итерационный for() имеет следующее формальное определение:

for ([инициализация счетчика]; [условие]; [изменение счетчика]){ // действия }

например фрагмент кода ниже:

for(int x = 10; x < 15; x = x+1) { System.out.print("Значение x: " + x ); System.out.print("\n"); }

Возможные виды изменения счетчика: i++, i--, ++i, --i, i+/- const (где i – переменнаясчетчик)

2) С предусловием while(), например фрагмент кода ниже: int x = 10; while( x < 15 ) { System.out.print("Значение x: " + x); x++; System.out.print("\n"); }

3) С постусловием do{} while(), например фрагмент кода ниже: int x = 10; do{ System.out.print("Значение х: " + x ); x++; System.out.print("\n"); } while( x < 15 );

4) Отдельно стоит оператор for : each для работы с массивами и коллекциями, например фрагмент кода ниже: int [] numbers = {-1, 0, 5, 6, 10}; for(int x : numbers ){ System.out.print(x); System.out.print("\n"); }

Использование циклов для работы с массивами

Перебор массива int[] aNums = { 2, 4, 6 }; for (int i = 0; i < aNums.length; i++) { System.out.println(aNums[i]); } или for (int i: aNums) { System.out.println(i); }

Одним из ключевых методов интерфейса Collection является метод Iterator iterator(). Он возвращает итератор - то есть объект, реализующий интерфейс Iterator.

Интерфейс Iterator имеет следующее определение: public interface Iterator { E next(); boolean hasNext(); void remove(); }

Реализация интерфейса предполагает, что с помощью вызова метода next() можно получить следующий элемент. С помощью метода hasNext() можно узнать, есть ли следующий элемент, и не достигнут ли конец коллекции. И если элементы еще имеются, то hasNext() вернет значение true. Метод hasNext() следует вызывать перед методом next(), так как при достижении конца коллекции метод next() выбрасывает исключение NoSuchElementException. И метод remove() удаляет текущий элемент, который был получен последним вызовом next().

Используем итератор для перебора коллекции ArrayList: import java.util.\*; public class Program { public static void main(String[] args) { ArrayList arr = new ArrayList<>(Arrays.asList(1, 3245, 3, 5, 7, 23)); Iterator iter = arr.iterator(); while(iter.hasNext()){ System.out.println(iter.next()); } } }

Интерфейс ListIterator расширяет интерфейс Iterator и определяет ряд дополнительных методов:

void add(E obj): вставляет объект obj перед элементом, который должен быть возвращен следующим вызовом next()

boolean hasNext(): возвращает true, если в коллекции имеется следующий элемент, иначе возвращает false

boolean hasPrevious(): возвращает true, если в коллекции имеется предыдущий элемент, иначе возвращает false

E next(): возвращает текущий элемент и переходит к следующему, если такого нет, то генерируется исключение NoSuchElementException

E previous(): возвращает текущий элемент и переходит к предыдущему, если такого нет, то генерируется исключение NoSuchElementException

int nextIndex(): возвращает индекс следующего элемента. Если такого нет, то возвращается размер списка

int previousIndex(): возвращает индекс предыдущего элемента. Если такого нет, то возвращается число -1

void remove(): удаляет текущий элемент из списка. Таким образом, этот метод должен быть вызван после методов next() или previous(), иначе будет сгенерировано исключение IlligalStateException

void set(E obj): присваивает текущему элементу, выбранному вызовом методов next() или previous(), ссылку на объект obj

Используем ListIterator: import java.util.\*; public class Program { public static void main(String[] args) { ArrayList arr = new ArrayList<>(Arrays.asList(1, 3245, 3, 5, 7, 23)); ListIterator listIter = arr.listIterator(); while(listIter.hasNext()){ System.out.println(listIter.next()); } // сейчас текущий элемент - 23 // изменим значение этого элемента listIter.set(66); // пройдемся по элементам в обратном порядке while(listIter.hasPrevious()){ System.out.println(listIter.previous()); } } }

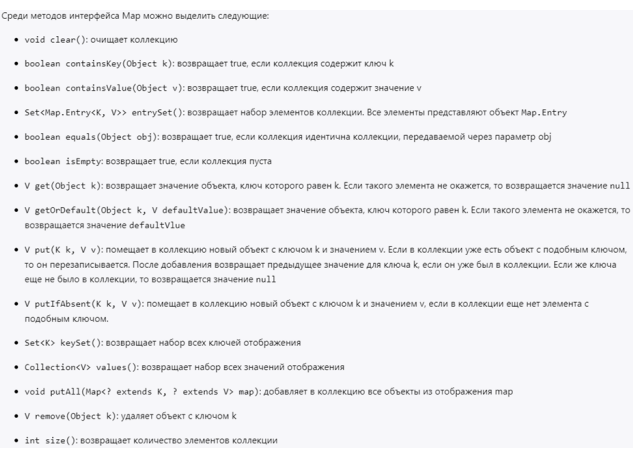
**Возможности Java Framework Collection. Интерфейс Map и его основные методы.**

Возможности Java Framework Collection

Java Collection Framework — иерархия интерфейсов и их реализаций, которая является частью JDK и позволяет разработчику пользоваться большим количеством структур данных из «коробки».

Интерфейс Map и его основные методы

Интерфейс Map представляет отображение или, иначе говоря, словарь, где каждый элемент представляет пару "ключ-значение". При этом все ключи уникальные в рамках объекта Map. Такие коллекции облегчают поиск элемента, если нам известен ключ - уникальный идентификатор объекта. В отличие от других интерфейсов коллекций не наследуется от интерфейса Collection (!!!).

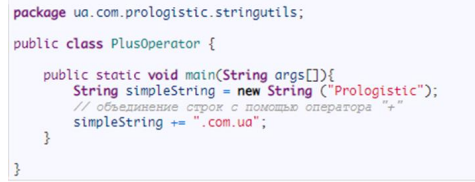


**Билет 21**

**Обработка строк в Java. Класс StringBuffer. Класс StringBuffer.**

При работе со строками, а именно конкатенацией, не желательно использовать оператор «+». Для этих целей следует использовать StringBuffer или StringBuilder.

С помощью “+”:



Конкатенация строк с помощью StringBuffer и StringBuilder будут аналогичны друг другу:



Объекты String являются неизменяемыми, поэтому все операции, которые изменяют строки, фактически приводят к созданию новой строки, что сказывается на производительности приложения. Для решения этой проблемы, чтобы работа со строками проходила с меньшими издержками в Java были добавлены классы StringBuffer и StringBuilder. По сути, они напоминает расширяемую строку, которую можно изменять без ущерба для производительности.

Эти классы похожи, практически двойники, они имеют одинаковые конструкторы, одни и те же методы, которые одинаково используются. Единственное их различие состоит в том, что класс StringBuffer синхронизированный и потокобезопасный. То есть класс StringBuffer удобнее использовать в многопоточных приложениях, где объект данного класса может меняться в различных потоках. Если же речь о многопоточных приложениях не идет, то лучше использовать класс StringBuilder, который не потокобезопасный, но при этом работает быстрее, чем StringBuffer в однопоточных приложениях.

Список методов, поддерживаемых классом StringBuffer:

№ Описание

1 public StringBuffer append(String s) Обновляет значение объекта, который вызывает метод. Этот метод принимает boolean, char, int, long, Strings и т.д.

2 public StringBuffer reverse() Этот метод изменяет значение объекта StringBuffer, который вызывает метод.

3 public delete(int start, int end) Удаляет строку, начиная с начального индекса до конечного индекса.

4 public insert(int offset, int i) Этот метод вставляет строку s в позицию упомянутую по смещению.

5 replace(int start, int end, String str) Этот метод заменяет символы в подстроке данного StringBuffer символами в указанной строке.

Паттерн

Паттерны проектирования (шаблоны проектирования) — это готовые к использованию решения часто возникающих в программировании задач. Это не класс и не библиотека, которую можно подключить к проекту, это нечто большее. Паттерны проектирования, подходящий под задачу, реализуется в каждом конкретном случае.

Типы паттернов: ● порождающие ● структурные ● поведенческие

Порождающие паттерны предоставляют механизмы инициализации, позволяя создавать объекты удобным способом.

Структурные паттерны определяют отношения между классами и объектами, позволяя им работать совместно.

Поведенческие паттерны используются для того, чтобы упростить взаимодействие между сущностями.

**Напишите универсальный класс для реализации алгоритмов поиска. В качества параметров используйте массив интерфейсных ссылок**

|  |
| --- |
| package ru.minusd.ticket21; |
|  |  |
|  | public class Searcher { |
|  | public static int search(Comparable[] array, Object obj) { |
|  | for (int i = 0; i < array.length; i++) { |
|  | if (array[i].compareTo(obj) == 0) { |
|  | return i; |
|  | } |
|  | } |
|  | return -1; |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | Comparable[] array = new Comparable[3]; |
|  | array[0] = new Cat("Murzik"); |
|  | array[1] = new Cat("Barsik"); |
|  | array[2] = new Cat("Vaska"); |
|  |  |
|  | System.out.println (search(array, new Cat("Murzik"))); |
|  | System.out.println (search(array, new Cat("Barsik"))); |
|  | System.out.println (search(array, new Cat("Vaska"))); |
|  | System.out.println (search(array, new Cat("Tom"))); |
|  | } |
|  | } |

**Билет 22**

**Наследование, виды наследования и его реализация в Java и C++**

Наследование в Джава — это механизм, который позволяет создавать новые классы на основе уже существующих. Новый класс называется подклассом (дочерним классом, производным классом), а существующий класс называется суперклассом (родительским классом, базовым классом).

Ключевое слово extends используется для наследования класса. Оно указывает, что класс, который объявляется после ключевого слова extends, является суперклассом для класса, который объявляется до ключевого слова extends.

Существует несколько видов наследования:

* Одиночное наследование (single inheritance). При этом класс может наследовать только один суперкласс. Синтаксис:

class SubClass extends SuperClass {

// тело класса

}

* Множественное наследование. В Java множественное наследование не поддерживается. Однако стоит упомянуть, что класс может реализовывать несколько интерфейсов.
* Иерархическое наследование. При этом класс может наследовать суперкласс, который в свою очередь может наследовать другой суперкласс и т.д. Синтаксис:

class SubClass extends SuperClass {

// тело класса

}

class SubSubClass extends SubClass {

// тело класса

}

**Паттерны проектирования программ. Паттерн Фабричный метод**

**Паттерны проектирования** (шаблоны проектирования) — это готовые к использованию решения часто возникающих в программировании задач. Это не класс и не библиотека, которую можно подключить к проекту, это нечто большее. Паттерны проектирования, подходящий под задачу, реализуется в каждом конкретном случае. Следует, помнить, что такой паттерн, будучи примененным неправильно или к неподходящей задаче, может принести немало проблем. Тем не менее, правильно примененный паттерн поможет решить задачу легко и просто.

**Типы паттернов**:

* *порождающие (фабрика тут)*
* *структурные*
* *поведенческие*

Порождающие паттерны предоставляют механизмы инициализации, позволяя создавать объекты удобным способом. Структурные паттерны определяют отношения между классами и объектами, позволяя им работать совместно. Поведенческие паттерны используются для того, чтобы упростить взаимодействие между сущностями.

Шаблон проектирования Фабрика позволяет управлять созданием объектов.

Билет 23

**Расширение классов. Порядок создания экземпляра дочернего класса.**

Пример: класс Cat - расширение класса Animal.

Расширением это называется, потому что Cat будет иметь все методы и поля класса Animal плюс в нем могут быть объявлены свои методы и поля, т.е. с точки зрения своего функционала класс станет шире. В связи с вышесказанным и требуется явное приведение. Представим, что у нас есть класс Animal и Cat его наследующий. В классе Animal есть один метод - к примеру, run();

Тогда мы можем объявить и проинициализировать переменную

Animal animal = **new** Animal()

после чего через переменную animal можем вызвать метод run()

animal.run()

Но также мы можем объявить и проинициализировать переменную следующим образом:

Animal cat = **new** Cat();

И вновь можем вызвать метод

cat.run();

так как Cat наследуется от Animal.

Теперь объявим в методе Cat метод, к примеру jump(); Мы можем создать переменную

Cat cat = **new** Cat();

и соответственно вызвать метод

cat.jump();

Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово extends, после которого идет имя базового класса. Для класса Employee базовым является Person, и поэтому класс Employee наследует все те же поля и методы, которые есть в классе Person.

Если в базовом классе определены конструкторы, то в конструкторе производного классы необходимо вызвать один из конструкторов базового класса с помощью ключевого слова super.

**Понятие сериализации и ее использование в ООП программах.**

Сериализация представляет процесс записи состояния объекта в поток, соответственно процесс извлечения или восстановления состояния объекта из потока называется десериализацией. Сериализация очень удобна, когда идет работа со сложными объектами.

Интерфейс Serializable

Сразу надо сказать, что сериализовать можно только те объекты, которые реализуют интерфейс Serializable. Этот интерфейс не определяет никаких методов, просто он служит указателем системе, что объект, реализующий его, может быть сериализован.

Сериализация (Serialization) — это процесс, который переводит объект в последовательность байтов, по которой затем его можно полностью восстановить. Зачем это нужно? Дело в том, при обычном выполнении программы максимальный срок жизни любого объекта известен — от запуска программы до ее окончания. Сериализация позволяет расширить эти рамки и «дать жизнь» объекту так же между запусками программы.

Сериализация. Класс ObjectOutputStream

Для сериализации объектов в поток используется класс ObjectOutputStream. Он записывает данные в поток.

Для создания объекта ObjectOutputStream в конструктор передается поток, в который производится запись:

1 ObjectOutputStream(OutputStream out)

Для записи данных ObjectOutputStream использует ряд методов, среди которых можно выделить следующие: ● void close(): закрывает поток ● void flush(): очищает буфер и сбрасывает его содержимое в выходной поток ● void write(byte[] buf): записывает в поток массив байтов ● void write(int val): записывает в поток один младший байт из val ● void writeBoolean(boolean val): записывает в поток значение boolean ● void writeByte(int val): записывает в поток один младший байт из val ● void writeChar(int val): записывает в поток значение типа char, представленное целочисленным значением ● void writeDouble(double val): записывает в поток значение типа double ● void writeFloat(float val): записывает в поток значение типа float ● void writeInt(int val): записывает целочисленное значение int ● void writeLong(long val): записывает значение типа long ● void writeShort(int val): записывает значение типа short ● void writeUTF(String str): записывает в поток строку в кодировке UTF-8 ● void writeObject(Object obj): записывает в поток отдельный.

**Билет 24**

**Расширение классов. Переопределение методов. Скрытие полей данных**.

**Переопределение метода** (англ. Method overriding) в объектно-ориентированном программировании — одна из возможностей языка программирования, позволяющая подклассу или дочернему классу обеспечивать специфическую реализацию метода, уже реализованного в одном из суперклассов или родительских классов.

Чтобы задать нужное нам поведение, мы сделали несколько вещей:

Создали в каждом классе-наследнике метод с таким же названием, как и у метода в родительском классе.

Сообщили компилятору, что мы не просто так назвали метод так же, как в классе-родителе: хотим переопределить его поведение. Для этого «сообщения» компилятору мы поставили над методом **аннотацию @Override** («переопределен»).  
Проставленная над методом аннотация @Override сообщает компилятору (да и читающим твой код программистам тоже): «Все ок, это не ошибка и не моя забывчивость. Я помню, что такой метод уже есть, и хочу переопределить его».

Написали нужную нам реализацию для каждого класса-потомка. Змея при вызове voice() должна шипеть, медведь — рычать и т.д.

В Java можно объявлять переменные в классе, методе, блоке или конструкторе. В зависимости от варианта объявления, у переменной могут быть разные области видимости в программе, а сама переменная может относиться к статическому или нестатическому типу в зависимости от того, где она объявлена.

Основываясь на этих критериях, можно разделить все переменные в Java на следующие типы: **локальные переменные, переменные экземпляра** и **переменные класса**. Затенение (shadowing) и сокрытие (hiding) переменных происходит, когда оказываются одноименными две переменные в разных областях видимости (в локальной и глобальной области видимости, в родительских и дочерних классах). При таком сценарии используется значение той переменной, которая находится во внутренней области видимости, так как она затемняет/скрывает значение той переменной, что находится во внешней области видимости.

**Паттерны проектирование программ. Паттерн Observer и модель MVC.**

Паттерны проектирования (шаблоны проектирования) — это готовые к использованию решения часто возникающих в программировании задач. Это не класс и не библиотека, которую можно подключить к проекту, это нечто большее. Паттерны проектирования, подходящий под задачу, реализуется в каждом конкретном случае. Паттерн Наблюдатель (Observer) относится к поведенческим (behavioral) паттернам проектирования. Основная цель данного паттерна — это определение зависимости один-комногим между объектами, чтобы при изменении состояния объекта, все зависимые от него объекты получали уведомление и обновлялись автоматически. MVC расшифровывается как «модель-представление-контроллер» (от англ. modelview-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.