**170620**

ООП программа - это совокупность объектов, которые взаимодействуют друг с другом путем передачи сообщений. У каждого объекта есть состояние и поведение. Состояние описывается значениями в наборе полей у класса, а поведение описывается реакцией на сообщения, которые посылают другие объекты, и эта реакция описывается в классах в виде методов.

Класс - синтаксическая единица, которая описывает структуру совокупности объектов, принадлежащих этому классу. Класс сам по себе тоже объект. Когда запускается программа создается объект класса. Запускаются объекты классов, а потом уже идет обращение к ним в случае необходимости для создания экземпляров класса. Загружается байт код создается объект класса, потом идет поиск точки входа main и он запускается.

Инкапсуляция объединение данных и кода в классе и закрытие доступа к ним с помощью модификаторов доступа. ООП позволил повысить уровень абстракции и позволил повторно использовать код за счет наследования и делегирования. Передача сообщений объекта другому объекту осуществляется благодаря наличию доступа к другому объекту т.е. ссылки на другой объект.

Полиморфизм: наследование, переопределение методов.

**Переменный в Java полиморфны: ссылочной переменной мы можем присвоить любой объект(экземпляр любого класса), лишь бы он обладал тем же типом что и переменная. То что хранится в переменной обладает определенным типом имеет соответствующий интерефейс и может отвечать на соответствующие запросы.**

ООП позволяет структурировать код для удобной работы с ним.

[Книги Тимоти Бадд] [Роберт С Мартин Чистый код] [Идеальный программист тот же автор]

Codingbat.com, HackerRank.com

[ДЗ] – 1 в файле.

160625[2]

Конструктор конструирует объекты, а не создает их. Их создает JVM. Каждый объект в джаве является монитором. У каждого объекта есть hashcod. При создании объекта отводится память потом идет обращение к конструктору суперкласса, потом срабатывает инициализатор родителя, потом конструктор родителя, потом инициализатор наследника и конструктор наследника. Инициализатор нужен, если у нас есть много разных конструкторов, но в них есть что-то общее. И это общее мы можем запихнуть в блок инициализации.

**Что отличает конструкторы от остальных методов? Отсутствие типа возвращаемого значения. Конструкторы не наследуются и не переопределяются.**

**[3]Конструкторы обладают всеми модификаторами доступа.**

Синглтон – это класс который гарантирует существование только одного своего экземпляра. Static переменные существуют в единственном числе.

Immutable объекты класса – это которые не изменяются (final) Можно безопасно использовать в потоках. При использовании final нет значения по умолчанию. Мы обязаны присвоить ему значение либо в конструкторе, либо в блоке инициализации. Поля инициализируются при создании классов значением по умолчанию. А локальные переменные нужно инициализировать.

Ссылочные типы – это адрес в динамической куче 4байта если 32 битная java машине, 64 в битной. 64 битные работают медленнее. Локальные переменные располагаются в стеке выполнения(массив из 32 или 64 бит ячеек). Туда помещаются переменные методов, которые выполняются в данный момент. Разница между ссылочными типами и примитивными типами состоит в том, что в примитивных хранится значение, а в адрес памяти в куче, где хранится объект.

**170627[06] ДЗ**

**170629 –** Синглтоный – Laizy initialization. В самом начале происходит инициализация полей класса и констант на этапе компиляции, потом статики. Объект(экземпляр) описывающий сам класс один, а потом создается множество экземпляров класса. Стек вызовов.

[03]-при компиляции вложенные кассы отделяются знаком доллара и именем класса. А анонимные классы просто номерами. Поэтому не рекомендуется использовать его в именах полей и т.д. Массив args существует он не будет null. Параметры в отличие от С++ не передаются. Методы экземпляра те где неявно присутствует ссылка на тот объект к которому обратились, и эта ссылка всегда называется this.

[4] ClassLoader – объет который загружает классы. С помощью наследования можно использовать повторный код. С помощью полиморфизма мы можем писать обобщенный код, который рабоет с произвольными объектами произвольных классов, главное, чтобы они обладали соответствующим типом. И за счет наследования мы можем менять реакцию(поведение) объектов на одни и те же сообщения. В каждом методе экземпляра класса неявно присутствует ссылка на тот объект, к которому идет обращение и эта ссылка называется this и она нам доступна.

Внутри классов могут быть: поля класса, поля экземпляра, методы класса, методы экземпляра, статический (класса) инициализатор, инициализатор экземпляра, вложенные(внутренние) классы, интерфейсы. В классе есть описание пакета, есть импорты для упрощения. Сокращения имен.

[05]

static void classMethod(){  
 synchronized (SyncExample.class){  
   
 }  
}  
 void instanceMethod(){  
 synchronized (this){  
  
 }  
}

Конструктор – это метод особого вида.

Class<A> c = A.class;

// A a = new A();  
  
 A a = c.newInstance();  
  
 Constructor<A> constructor = c.getConstructor(null);  
 constructor.newInstance(null);

// a.m(); эквивалент  
 m.invoke(a,null);

У каждой переменной есть время жизни Variable Scope. Hiding – некоторые сущности называются одинаково и закрывают некоторый объект соблой.

**170701[1]** – Программа геном. {Дз доделать прогу и сделать с хэш таблицами}

**170701[2]- операции.**

**[03] - (Java Puzzlers, Седжвик книги)**

**[04]GuardConditions**

**[06] Матрица в джава – массив массивов. Break OUTER. {ДЗшка}**

**170704[01]**

Можно ли из статического метода обратиться к переменному экземпляру? Ответ: можно, только если есть ссылка, т.е. создать объект класса, а потом уже работать с внутренними переменными и методами. При печати ссылочного типа данных печатается не ссылка, хэш-код, вычисленный на основе адреса.

{ДЗ}

[2] – инициализация массивов.

[4] – класс Arrays. Для чего нужны сортировки? Ответ: это красиво=)), двоичный поиск скорость logn, min max, поиск дубликатов. Хеширование будет работать дольше. Как отсортировать 1000000 целых чисел, наверно quicksort – неверно. RadixSort[Курс Седжвика]LSD, NSD. Сортировка поразрядная скорость линейная

RADIX: 2\*w\*N, где 2\*4\*1000000 = 80000000 Поразрядная сортировка. {ДЗ проверить это}

QuickSort: 1.4NLogN 280000000

[05] – vararg ArrayList – динамический массив – неограниченный размер, можно изменять на лету.

170706

[02] – Лучше для накопления и добавления данных к строкам использовать StringBuilder and String buffer. Так как при конкатенации и добавлении новых элементов в массив копируется весь массив в новый и возвращается. StringBuilder – небезопасный с точки зрения многопоточности, StringBuffer – безопасный с точки зрения многопоточности. Если одноразовая операция, то можно и конкатенацию можно использовать, а если у нас много элементов и там нужно с ними работать, то использовать StringBuilder.

[03]Строковые литералы, если они одинаковые создаются один раз и потом используются (Создается один объект строкового литерала). Литералы помещаются в специальном хранилище. Intern String Pool Size хранилище с каноническим представлением.

String intern

Строки immutable - неизменяемые, т.е. каждый раз когда мы что-нибудь делаем со строками, генерируется новая строка.

170707

Строки продолжение. Обзор методов для работы со строчками.

[01] Перегрузка методов – это одинаковые названия, но разные параметры. Приведение к каноническому виду. {ДЗ применить TDD и реализовать тест для }

[04] – продолжение c ArrayList. Из списка удаляется объект, ткоторый равен тому, который мы указали. Могут быть объекты с одинаковым содержимым и, если он по содержимому схож с тем элементом, который лежит в списке. Можно ли быстро проинициализировать ArrayList? Да, можно

ArrayList list = new ArrayList(){{  
 add("one");  
 add("two");  
 add("three");  
}};

List<String> list2 = Arrays.*asList*("one","two","three");

ArrayList<String> list2 = (ArrayList)Arrays.*asList*("one","two","three");

Нельзя добавлять элементы, удалять, но можно изменять.

List<String> list2 = Arrays.*asList*("one","two","three");

Неизменяемый список:

List<String> unmodifiableList = Collections.*unmodifiableList*(list);

Wrapper - обертка

*unmodifiableList*(list)

[04] iluwatar/java-design-patterns

[05] создание своего ArrayList

{ДЗ}

170710

Объект может создаваться, мутировать (изменять свое состояние), можно спросить каково его состояние.

[02] Создание LinkedList {ДЗ сделать вместо класса массив с объектами}

Класс, в котором только поля, конструктор, геттеры и сеттеры. Фактически структура без поведения.

*//POJO  
//Plain Old Java Project*

Travers – обход чего-либо.

[03] {ДЗ} сделать с помощью stringBuilder. Написать remove для индекса.

[04] – wrapper classes. Обертки. Hashtable – ассоциативная таблица(массив)

table.put(**new** Integer(10000),**"один"**); **wrapper** deprecated – делают объекты из примитивных типов.  
  
[05]table.put(2,**"two"**); **Autoboxing** автоупаковка, **Unboxing** распаковка.

[06] – Время сейчас используется:

System.***out***.println(LocalTime.*now*());  
System.***out***.println(LocalDate.*now*());  
System.***out***.println(LocalDateTime.*now*());

Фреймворк для работы с временем и производительностью. JMH.

170712

Книга: «Карьера программиста» Г.Лакман Макдауэлл 6-е издание. {ДЗ – 30 days of coding|| Cracking the Coding Interview}

[01]-Stack магазин – LIFO Last In First Out: push, pop, tos(taps of stack), top, peek.

Queue FIFO

[03] {ДЗ применить TDD к написанию стека}

[04] – ООП в java. В ссылочных переменных хранятся адреса объектов. Пересылка сообщений – это вызов метода у класса. Объекты обмениваются друг с другом путем передачи сообщений. Это некая сущность, которая может реагировать на поступающие из вне сообщения и менять свое состояние (мутировать) и генерировать новые события и передавать их другим из вне известным этому объекту сущностям. Объект - Совокупность переменных и методов, связанных с этими переменными.

[05] Инкапсуляция. Поля и методы, объединенные в классе это первая часть инкапсуляции.

1963,12,17 Др Заала

При переопределении методов класса, мы можем оставить тот же модификатор доступа, что и у родителя или поставить более слабый (раскрыть доступ, но никак не закрыть).

170714

[02]

Class<A> c = A.**class**; Получаем доступ к объекту описывающему класс А. Динамическое связывание. Все операции в джава идут по значению, мы копируем значение для примитивных типов, а для ссылочных копируем адрес ссылки.

[03] Явное нисходящее преобразование типов от родителя к потомку.

o = **new** A();  
a= (A) o;

Восходящее преобразование от потомка к родителю

A a;  
a = **new** A();  
a = **new** B();

a = e;  
e = (E) a;  
((E)a).m(); Касты можно использовать, но тогда со структуруй что-то не то. Instanceof() – тоже признак, что что-то не в порядке.

**Полиморфность переменных – в переменные можно сохрать ссылки любых объкектов, лишь бы они обладали определенным типом. Наследование в качестве переопределения методов у родителя. Механизмы обеспечивающие полиморфизм: наследование, полиморфность переменных и переопределение методов.**

Так лучще не делать:

Pet pet = **new** Dog();  
pet.feed();  
Dog dog = (Dog) pet;  
**if** (pet **instanceof** Cat){  
Cat cat = (Cat) pet;   
}

[04] Абстрактные классы

Может ли абстрактный класс не содержать методов вообще? Ответ: да, может. Может ли абстрактный класс содержать не только абстрактные методы? Ответ: да может. Там должны быть только абстрактные методы? Нет могут быть и с реализацией.

Метод родителя можно вызвать не только из переопределенного метода, но и из любого другого и в любом месте класса, когда угодно. Методы – это тоже объекты.

**super**.makesound(); В объекте описывающем класс родителя, найти метод makesound, выбрать его код и применить его.

Можно ли сделать super.suoer.makesound() Так делать нельзя. Мы можем обраться только к медоду родителя ближайшего к нам. Если в ближайшем классе родителя нет этого метода, то ищется дальше.

Super говорит о том, что мы идем вверх по иерархии наследования и ближайшую реализацию, которую мы обнаружили, м ы используем.

[05]-Статическое и динамическое связывание. HIDING Final в применении к классам или методам. В момент вызова метода нам известно куда перепрыгнуть для выполнения следующего кода. Динамическое применяется в наследовании. Если класс final, то от него нельзя унаследоваться. И если метод тоже final, то его нельзя переопределить.

Статические методы вызываем при обращении к имени класса. Final в применении к классу, запрещает его наследовать, при примении к методу хапрещает его переопределять, при применении к переменной запрещает ее изменять. Передача параметров по значению или по ссылке. В джаве всегда происходит передача параметров по значению. Передача по значению – это значит, что значение параметра переменной будет скопировано в локальную переменную. Final можно еще применить к параметру метода.

170717{solitaire}

[1] книги Sedgevik, Simple Java, Статистика и котики.

[04] – Solitaire. Метод мутатор, который меняет внутреннее состояние объекта.

170719

[01]- Swap

x=x+y; *//x==100 + 50 ==150*y=x-y; *//y=100*x=x-y; *//x = 50*

В джаве все параметры передаются по значению. Т.е. то что мы передаем в качестве параметров в метод, копируется в локальные переменные.

[02] Виды наследования:

SOLID

Specialization is-a Liskov Substitution Principle

Specification is-a

Construction NO is-a

Generalization is-a?

Extension

Restriction No is-a!

Variation No is-a!

Combination not implemented in Java for classes

[03] Конструкторы не наследуются. Рассматриваются верхние виды наследования.

[05] Начало интерфейсов!!!. Тип = Интерфейсы || Классы

170724

[01] – Интерфейсы метки – это пустые интерфейсы без методов. Класс адаптер используется для того, чтобы не реализовывать все методы, которые есть в интерфейсе. Интерфейсы могут наследовать друг друга.

[02] **Сериализация** – можно сделать из объекта последовательность байт. Можно делать копию объекта с помощью Serializable или с помощью Clonable.

[03] **Вложенные классы Nested(вложенный): Static, Inner, Local, Anonymous. SILA**

Inner вложенный

**public class** StaticExamples {  
 **static class** A **extends** StaticExamples{  
 **static class** B **extends** A{  
   
 }  
 }  
}

Вложенные классы нужны, чтобы не раскрывать особенности реализации во вне.

Создание объекта вложенного класса

StaticExamples.X x = **new** StaticExamples.X();

Статический класс – это класс, который просто физически располагается в другом классе.

[04] Интерфейсы поддерживают множественное наследование. А классы могут только множественно реализовывать интерфейсы.

**public interface** Brain **extends** Pet,Temp {  
}

**public class** Brain **implements** Pet,Temp {  
}

[05]

{ДЗ}

Зачем нужны локальные классы, если есть анонимные? Мы можем создать сколько угодно экземпляров анонимных классов, в то время как анонимные мы можем создать только один раз, а дальше это уже будут другие классы, даже если код повторяется.

170726

[01] !!! – Тесты oracle.

A a2 = **new** B();  
a2.*helper*();  
a2.change();

Чем отличаются интерфейсы от абстрактных классов? Ответ: Общего у них - типы. Абстрактный класс может иметь методы, которые реализацию, а интерфейс не может. И статический дефолтный метод это не тоже самое, что реализация метода в классе. В абстрактном классе могут быть приватные и protected методы, а в интерфейсе нет. В абстрактном классе могут быть описаны состояния в виде полей. В интерфейсе нет. То, что мы описываем в абстрактном классе, присутствует в экземпляре класса наследника. Это поля. Абстрактный класс может быть вложенным, а интерфейс не совсем.

[02] Фабричный метод. Локальным переменным нельзя задать модификатор private.

Вопрос, что можно сказать про строчку:

Private final Blacksmith blacksmith;

Это поле. Final присвоить значение можно только один раз. С большой буквы, то это тип поля. Следовательно, переменная ссылочного типа. Значение у этого поля по умолчанию null. И ей обязательно должно быть присвоено значение либо в инициализаторе экземпляра, либо в конструкторе класса. В наследниках ее нельзя проинициализировать.

[04] - итераторы

[05] Дефолтные методы в интерфейсах. В интерфейсах неявно добавляются модификаторы public и abstract. А в полях static. В интерфейсах могут быть статические поля. Фабричные методы в интерфейсах. Имена классов в системе. В интерфейсах можно описать классы и они static inner. И доступ к ним:

Special.A a = **new** Special.A();

Для полей по умолчанию добавляется public static final.

170728

[01] – композиция = дилигирование. 4й фундаментальный принцип ООП. Boilerplate code.

[02]Exceptions

[03] Бывают проверяемые и не проверяемые. Try Catch – это затратная операция, так что лучше использовать условие.

[04] EmptyList используют для передачи в любой код, который работает со списками и ничего не произодйдет. А если бы мы использовали null, то была бы исключительная ситуация. Дизайн паттерн Null Object. Вместо того, чтобы отправлять куда – то null, лучше отправить пустой объект. Стек вызовов вниз(в глубину) и вверх.

[02]Книга по тестам oracle: OCA/OCP Java SE 8 Programmer Practice Tests.

Исключительная ситуация – это объект. Exceptions. Можно использовать несколько боков catch и перехватывать сначала более узкоспециализированные исключения, а затем более широкие. Multicatch.

[03]ReThrow повторное выбрасывание. Перехватываем, полняем еще инфой и выбрасываем дальше.

[04]Try Finally.

[05] При работе с ресурсами использовать try finally внутри блока try catch.