Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

Дисциплина “Программирование”

Отчёт по лабораторной работе №3

Вариант №29465

Выполнил:

Ануфриев Андрей Сергеевич,

P3119

Проверил:

Ермаков Михаил Константинович

г. Санкт-Петербург

2024 год

Оглавление

Задание

Этапы выполнения работы:

1. Получить вариант
2. Нарисовать UML-диаграмму, представляющую классы и интерфейсы объектной модели и их взаимосвязи;
3. Придумать сценарий, содержащий действия персонажей, аналогичные приведенным в исходном тексте;
4. Согласовать диаграмму классов и сценарий с преподавателем;
5. Написать программу на языке Java, реализующую разработанные объектную модель и сценарий взаимодействия и изменения состояния объектов. При запуске программа должна проигрывать сценарий и выводить в стандартный вывод текст, отражающий изменение состояния объектов, приблизительно напоминающий исходный текст полученного отрывка.
6. Продемонстрировать выполнение программы на сервере `helios`.
7. Ответить на контрольные вопросы и выполнить дополнительное задание.

Текст, выводящийся в результате выполнения программы не обязан дословно повторять текст, полученный в исходном задании. Также не обязательно реализовывать грамматическое согласование форм и падежей слов выводимого текста.

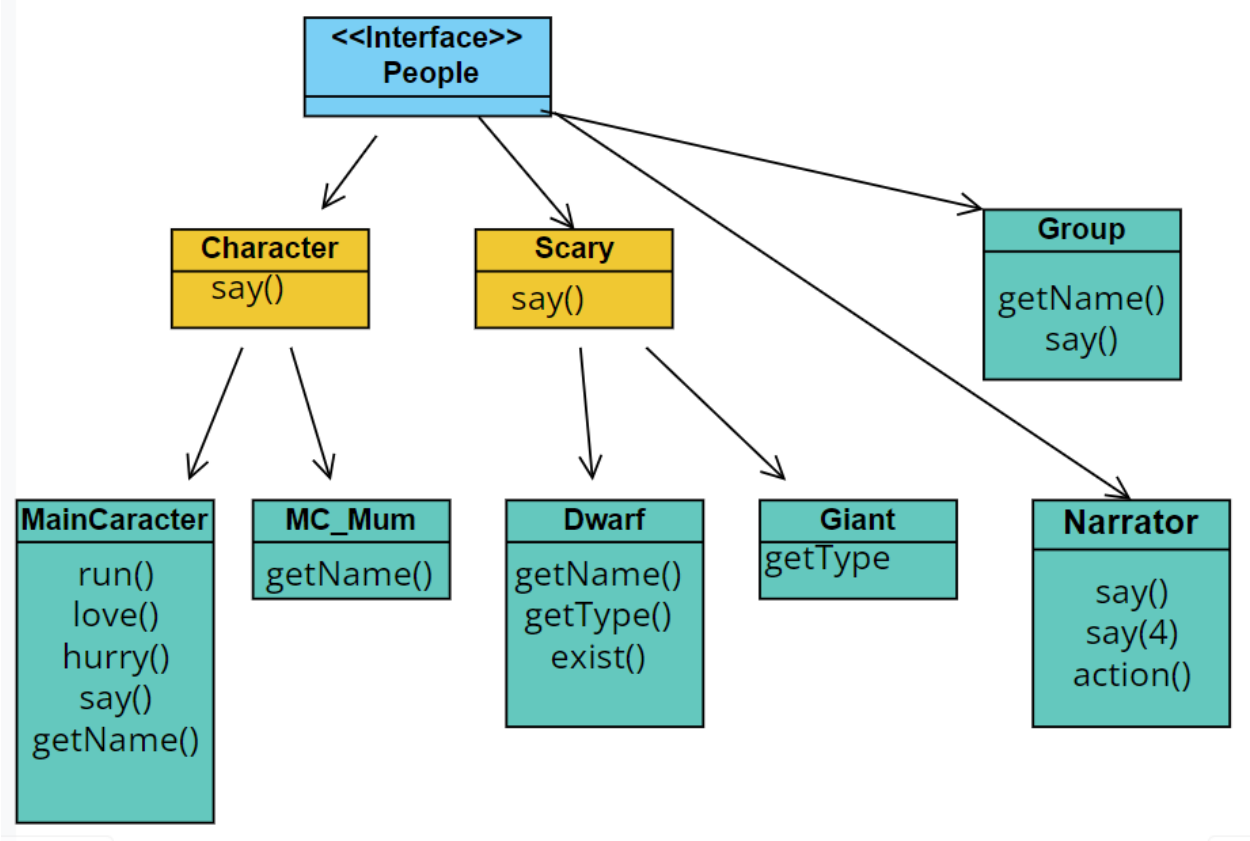
Стоит отметить, что цель разработки объектной модели состоит не в выводе текста, а в эмуляции объектов предметной области, а именно их состояния (поля) и поведения (методы). Методы в разработанных классах должны изменять состояние объектов, а выводимый текст должен являться побочным эффектом, отражающим эти изменения.

Требования к объектной модели, сценарию и программе:

1. В модели должны быть представлены основные персонажи и предметы, описанные в исходном тексте. Они должны иметь необходимые атрибуты и характеристики (состояние) и уметь выполнять свойственные им действия (поведение), а также должны образовывать корректную иерархию наследования классов.
2. Объектная модель должна реализовывать основные принципе ООП - инкапсуляцию, наследование и полиморфизм. Модель должна соответствовать принципам SOLID, быть расширяемой без глобального изменения структуры модели.
3. Сценарий должен быть вариативным, то есть при изменении начальных характеристик персонажей, предметов или окружающей среды, их действия могут изменяться и отклоняться от базового сценария, приведенного в исходном тексте. Кроме того, сценарий должен поддерживать элементы случайности (при генерации персонажей, при задании исходного состояния, при выполнении методов).

4. Объектная модель должна содержать как минимум один корректно использованный элемент каждого типа из списка:
    - абстрактный класс как минимум с одним абстрактным методом;
    - интерфейс;
    - перечисление (enum);
    - запись (record);
    - массив или ArrayList для хранения однотипных объектов;
    - проверяемое исключение.
5. В созданных классах основных персонажей и предметов должны быть корректно переопределены методы `equals()`, `hashCode()` и `toString()`. Для классов-исключений необходимо переопределить метод `getMessage()`.
6. Созданные в программе классы-исключения должны быть использованы и обработаны. Кроме того, должно быть использовано и обработано хотя бы одно unchecked исключение (можно свое, можно из стандартной библиотеки).
7. При необходимости можно добавить внутренние, локальные и анонимные классы.

Диаграмма классов



Исходный код программы

Андрей Ануфриев / Лаборатория 3 · ГитЛаб

Результат работы программы.

Пока Яков доедал последние ложки лакомого блюда , морские свинки зажгли аравийский ладан , комната наполнилась серый дымом.

Яков ест.

дым становился гуще и гуще , А запах ладана усыпительно действовал на мальчик.

Несколько раз он вспоминал, что ему пора вернуться к она , но вслед за тем его снова одолевала сильная дремота - он забывался и крепко заснул на диване у старуха.

Яков Яков спит.

ему мерещились странные сны.

ему казалось будто старуха снимает с него платье и одевала в бельчью шкуру.

он жил вместе с белка и морские свинки, которые оказались очень благовоспитанными особами, и вместе с ними прислуживал старуха.

Сначала ему поручали только чистку сапог, он должен был натирать до блеска маслом кокосовые скорлупки, служившие старухе туфлями.

Прошёл 1 год.

ему стали поручать более тонкую работу.

Вместе с несколькими другими белка он должен был ловить и собирать пылинки , А потом просеивать их сквозь тончайшее волосяное сито.

Дело в том, что старуха считала пылинки питательными веществами А так как она не имела зубов и не могла разжевать ничего твердого то ей пекли хлеб исключительно из пылинки.

Прошёл ещё год.

Яков был переведен в разряд слуг , которые собирали воду для питья старуха.

белка и Яков должны были собирать в ореховые скорлупки росу с роз.

старуха пьёт только такую воду поэтому у водоносов, в том числе и у Яков работа была не лёгкая.

Прошл ещё год.

Якова перевели на домашние работы.

ему поручено было содержать В чистоте пол , но так как он был стеклянным.

Яков должен был обертывать ноги старым сукном и разъезжать таким образом по всем комнатам.

На пятый год Якова перевели на кухню.

Яков прошел все степени: поваренка, помощника, первого повара , и достиг такой ловкости и уменья во всем, что часто дивился самому себе..

Так прошло 7 лет , Яков Яков служит старуха.

но вот однажды она сняла кокосовые туфли и, взяв в руку корзину , собралась уходить.

Она приказала Якову, чтобы к ей возвращению он ощипал курица , он начинил её зеленью и и хорошенька зажарил.

он свернул курица шею Яков сварил её он ощипал перья соскоблил кожу вынул из курицы внутренности.

В кладовой он увидел шкафчик , он заглянул туда В шкафу стояло много корзинок, от которых исходил приятный запах.

Яков открыл одну из корзинок и нашел в ней растение особенной формы и цвета.

Стебли и листья его были голубовато-зеленые, а цветок огненно-красный, с желтой каймой.

Яков задумчиво посмотрел на этот корзину понюхал его и Яков вспомнил, что он так же сильно пахнет, как тот суп, которым когда-то угостила его старуха.

Яков начал так сильно чихать , что проснулся.

Яков бодорствует.

он лежал на диване и осматривался кругом.

Удивительно, как можно видеть такие сны-

сказал он самому себе,

и притом с такой ясностью!

Вот ужо посмеется маменька, когда я ей расскажу все это!

С этими мыслями Яков поднялся с места, чтобы уйти Домой но всё тело окаменело от сна он не мог повернуть голова.

он невольно рассмеялся над собой и своей сонливостью , так как каждую минуту стукался нос то о шкаф, то о стену, то о косяк двери.

белка и морские свинки с визгом бегали вокруг.

улица куда привела его старуха , находилась в очень отдаленной части город.

Там была страшная толкотня!

По всей вероятности , думал он, где-нибудь поблизости показывают карлик так как он поминутно слышал возгласы:

-Ах , посмотрите на безобразного карлика!

Откуда он взялся?

Какой у него длинный нос и как смешно голова торчит у него прямо на плечах!

А руки - то , руки какие у него черный , безобразные!

В другое время он побежал бы за толпа , потому что очень он любил смотреть на великан , карлик и на всякие диковинки , но на раз ему было не до того : он спешил вернуться к она.

ему стало как-то жутко, когда он пришел на рынок.

мать все еще сидела на своем месте , потому что он проспал недолго.

Яков подошёл к ей сзади , ласково опустил руки на её плечах и сказал:

- Что с тобой, мамочка, ты сердишься на меня?

мать обернулась, но в ту же минуту отшатнулась от него с криком ужаса.

Что тебе нужно от меня, безобразный карлик ! – воскликнула она

- Прочь, прочь от меня, я терпеть не могу подобных шуток!

Но, мамочка, что с тобой ? спросил Яков с испугом.

- Тебе, верно, нездоровится , Зачем же ты гонишь меня, своего сына?

Я уже сказала тебе: убирайся прочь ! сказал она с гневом.

- От меня ты не получишь ни гроша за свои шутки, уродливое создание!

Вот горе-то, она совсем помешалась ! - подумал огорчённый Яков.

Как бы мне отвести ее домой?

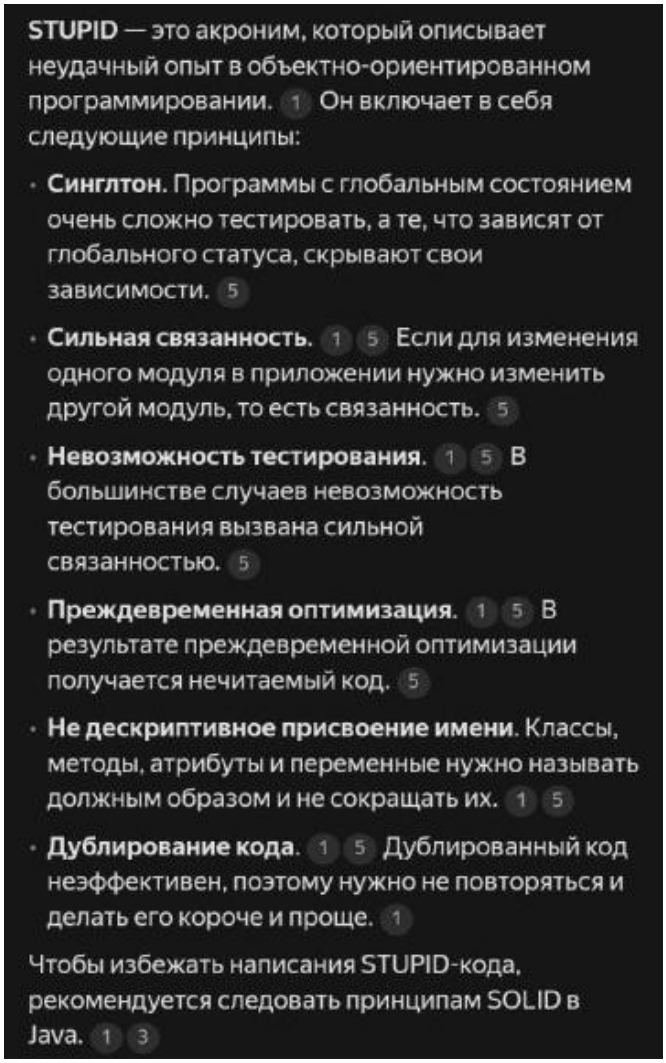
Вывод

Я научился проектировать и реализовывать классы. Узнал и применил основные принципы ООП в Java.

Ответы на вопросы

1. Принципы объектно-ориентированного программирования SOLID и STUPID.

- Принцип единственной ответственности
  Модуль должен иметь только одну причину для изменения
  Модуль должен быть ответственным только за одного актора.
- Принцип открытости/закрытости
  Элемент ПО должен быть открыт для расширения, но закрыт для изменения
- Принцип подстановки Барбары Лисков
  Подклассы должны подставляться на место их базового класса
- Принцип разделения интерфейса
  Много специализированных интерфейсов лучше одного универсального
- Принцип инверсии зависимостей
  Модули верхнего уровня не должны зависеть от модулей нижнего уровня. И те, и другие должны зависеть от абстракций. Абстракции не должны зависеть от реализации. Реализации должны зависеть от абстракций.



STUPID — это акроним, который описывает неудачный опыт в объектно-ориентированном программировании. 1 Он включает в себя следующие принципы:

- **Синглтон.** Программы с глобальным состоянием очень сложно тестировать, а те, что зависят от глобального статуса, скрывают свои зависимости. 5
- **Сильная связанность.** 1 5 Если для изменения одного модуля в приложении нужно изменить другой модуль, то есть связанность. 5
- **Невозможность тестирования.** 1 5 В большинстве случаев невозможность тестирования вызвана сильной связанностью. 5
- **Преждевременная оптимизация.** 1 5 В результате преждевременной оптимизации получается нечитаемый код. 5
- **Не дескриптивное присвоение имени**. Классы, методы, атрибуты и переменные нужно называть должным образом и не сокращать их. 1 5
- **Дублирование кода.** 1 5 Дублированный код неэффективен, поэтому нужно не повторяться и делать его короче и проще. 1

Чтобы избежать написания STUPID-кода, рекомендуется следовать принципам SOLID в Java. 1 3

2. Класс `Object`. Реализация его методов по умолчанию.

Класс Object лежит в основе всей иерархии классов Java

public native int hashCode()	**возвращает хэш-код объекта**
public boolean equals(Object obj)	**сравнивает объекты на равенство**
public native Object clone()	возвращает копию объекта
public String toString()	**преобразует объект в строку символов**

forName(String className)	возвращает объект Class для заданного имени
getName()	возвращает имя класса
newInstance()	создает новый экземпляр класса
getSuperclass()	возвращает суперкласс
isInterface()	определяет, является ли объект интерфейсом
getInterfaces()	возвращает интерфейсы класса
isArray()	определяет, является ли объект массивом
isPrimitive()	определяет, является ли тип примитивным

- **Метод equals()**. По умолчанию сравнивает объекты по ссылке. Однако в подклассах этот метод может быть переопределён для сравнения содержимого объектов.
- **Метод hashCode()**. По умолчанию возвращает номер ячейки памяти, где объект сохраняется.
- **Метод toString()**. По умолчанию возвращает строку, содержащую имя класса и хеш-код объекта. Однако этот метод также может быть переопределён для предоставления более информативного представления объекта

3. Простое и множественное наследование. Особенности реализации наследования в Java.

Простое наследование в Java — когда один подкласс наследует свойства и методы только у одного суперкласса. Для реализации этого типа наследование используется ключевое слово **extends**. 12
Множественное наследование в Java — когда один подкласс может наследовать свойства и методы сразу от нескольких суперклассов. 23 Для этого можно использовать **интерфейсы**. 24 Интерфейс в Java представляет собой абстрактный тип данных, который определяет набор методов без их конкретной реализации. Классы могут реализовывать один или несколько интерфейсов, что позволяет им наследовать функциональность от нескольких источников. 2

В Java нельзя наследоваться от следующих объектов:

- **Приватные переменные и методы**. Они не доступны классу-наследнику. 2
- **Статические методы**. Если в родительском классе объявлен статический метод, то дочерние классы его не наследуют. 3
- **Конструкторы и инициализаторы**. Они не наследуются, поскольку не являются членами класса, а служат для его инициализации. 1

Также в Java **нельзя наследовать самого себя**

4. Понятие абстрактного класса. Модификатор `abstract`.

Абстрактный класс в Java — это класс, экземпляр которого **нельзя создать сам по себе** — он служит основой для других классов. 5 В абстрактном классе также можно определить поля и методы, но в то же время нельзя создать объект или экземпляр абстрактного класса. 3
Модификатор abstract в Java используется для указания, что класс или метод **не имеют реализации** и должны быть реализованы в подклассе. 1
Некоторые особенности абстрактных классов:

- **Могут содержать как абстрактные, так и обычные методы**. 1
- **Могут использоваться в качестве базового класса** для других классов, которые реализуют их абстрактные методы.

5. Понятие интерфейса. Реализация интерфейсов в Java.
   Отличие интерфейсов от абстрактных классов.

1. **Интерфейс описывает только поведение. У него нет состояния. А у абстрактного класса состояние есть: он описывает и то, и другое.**
2. **Абстрактный класс связывает между собой и объединяет классы, имеющие очень близкую связь. В то же время, один и тот же интерфейс могут реализовать классы, у которых вообще нет ничего общего.**
3. **Классы могут реализовывать сколько угодно интерфейсов, но наследоваться можно только от одного класса.**

6. Модификаторы `default`, `static` и `private` для методов интерфейса.

Некоторые модификаторы для методов интерфейса в Java:

- **public**. 23 Относится только к интерфейсам верхнего уровня и интерфейсам участников, но не к локальным интерфейсам. 3
- **protected** и **private**. Относятся только к интерфейсам участников. 3
- **static**. Относится только к интерфейсам-членам и локальным интерфейсам. 3

Если в интерфейсе определяется метод, но не указывается модификатор доступа, то он автоматически считается **public** и **abstract**. 2
Также, начиная с Java 8, интерфейсы могут иметь **методы по умолчанию** (default methods), которые имеют реализации по умолчанию и могут быть переопределены в классах, реализующих интерфейс. 2

7. Перечисляемый тип данных (enum) в Java. Особенности реализации и использования.

Перечисляемый тип данных (enum) в Java представляет собой набор логически связанных констант. 1 Он используется для создания типов, которые могут иметь только несколько возможных значений. 2

Особенности реализации enum:

- Объявление происходит с помощью оператора enum, после которого идёт название перечисления. Затем идёт список элементов перечисления через запятую. 1
- Константы перечисления являются static final и не могут быть изменены после создания. 3
- Перечисления, как и обычные классы, могут определять конструкторы, поля и методы. 1

Использование enum:

- Перечисления позволяют определить набор значений, которые может принимать переменная. 5
- Использование enum помогает избежать ошибок, связанных с применением недопустимых значений. 5
- Enum можно использовать для реализации паттернов проектирования singleton (одиночка) и strategy (стратегия). 5

Некоторые методы enum:

- **name()** — конечный метод, который возвращает значение константы; 5
- **toString()** — также возвращает значение константы, но может быть переопределён; 5
- **ordinal()** — возвращает позицию константы, начиная с нуля; 5
- **valueOf(String)** — создаёт перечисление из строкового значения; 5
- **values()** — возвращает массив со значениями перечисления. 5

8. Тип запись (record) в Java. Особенности использования.

Тип записи (record) в Java — это класс, предназначенный для хранения данных.

Особенности использования записи:

По умолчанию записи неизменяемы. Это означает, что их состояние не может быть изменено после их создания. 2

9. Методы и поля с модификаторами `static` и `final`.
10. Перегрузка и переопределение методов.

Перегрузка метода означает создание другого метода с тем же именем в том же классе, но с другим списком параметров.

Ключевые правила перегрузки методов

- Перегруженные и перегружаемые методы должны находиться в одном классе (Примечание: сюда относятся любые методы, унаследованные, даже неявно, от суперкласса).
- Параметры метода должны измениться: либо количество, либо тип параметров должны быть разными в двух методах.
- Тип возвращаемого значения может быть свободно изменен.
- Модификатор доступа (`public`, `private` и другие) можно свободно изменять.
- Выброшенные исключения, если таковые имеются, могут быть свободно изменены.

Переопределение метода — это возможность реализации метода в подклассе, который уже существует в суперклассе или родительском классе.

- Список параметров не должен меняться: переопределяющий метод должен принимать то же количество и тип параметров, что и переопределяемый метод, иначе вы просто перегрузите метод.
- Тип возвращаемого значения не должен изменяться (Примечание: если метод возвращает объект, то в качестве типа возвращаемого значения допускается подкласс этого объекта).
- Модификатор доступа должен быть таким же или менее ограничивающим (например, если переопределяемый метод — `protected`, вы можете объявить переопределяющий метод как `public`, но не `private`).
- Выброшенные проверенные исключения, если таковые имеются, могут быть удалены или сокращены методом переопределения. Это означает, что переопределяющий метод может генерировать то же проверенное исключение, что и переопределенный метод, или подкласс этого проверенного исключения, но не более широкое исключение. Это ограничение не распространяется на непроверенные исключения.

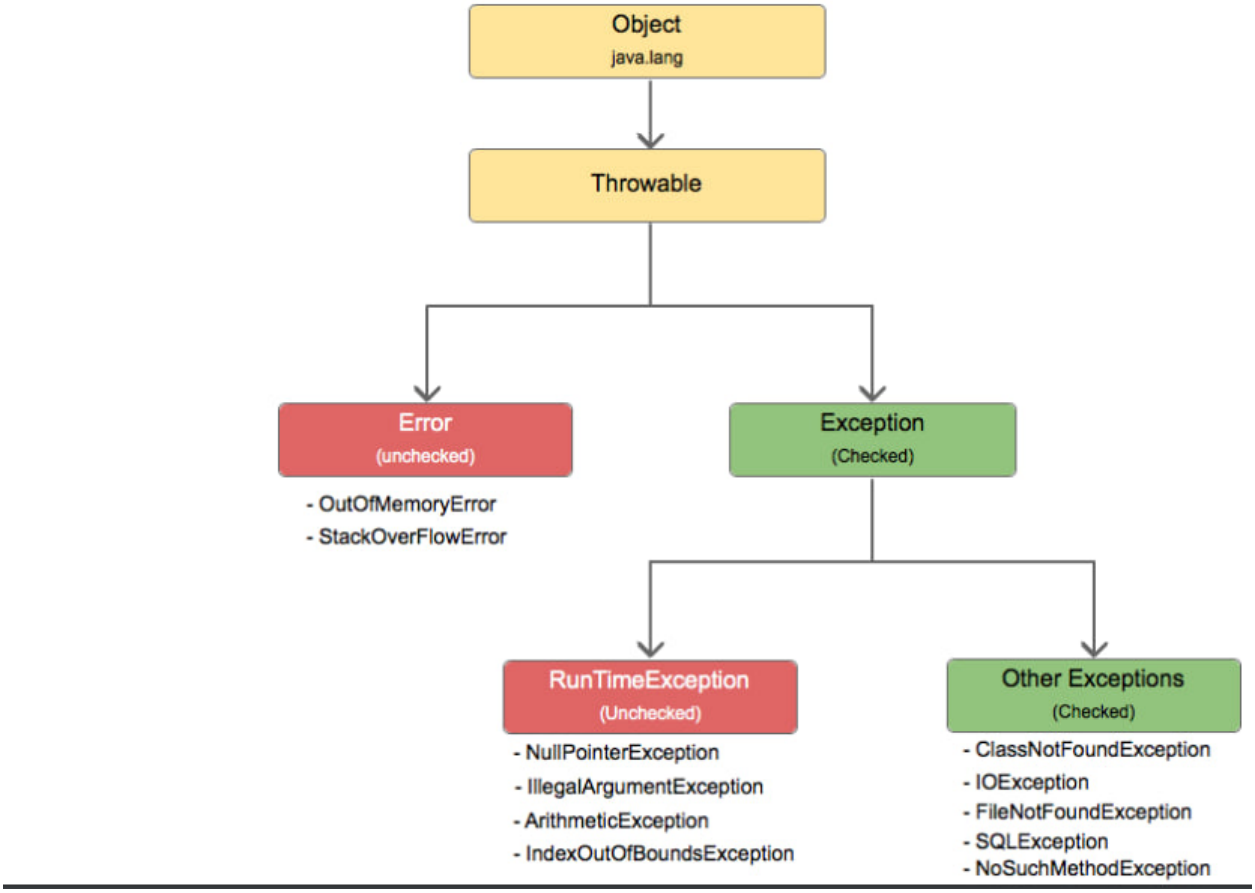
11. Обработка исключительных ситуаций, три типа исключений.

- try – определяет блок кода, в котором может произойти исключение;
- catch – определяет блок кода, в котором происходит обработка исключения;
- finally – определяет блок кода, который является необязательным, но при его наличии выполняется в любом случае независимо от результатов выполнения блока try.

Эти ключевые слова используются для создания в программном коде специальных обрабатывающих конструкций: try{}catch, try{}catch{}finally, try{}finally{}.

- throw – используется для возбуждения исключения;
- throws – используется в сигнатуре методов для предупреждения, о том что метод может выбросить исключение.

Исключения и их классификация



12. Стандартный массив и динамический массив (ArrayList). Основные различия.

Основные различия между стандартным массивом и динамическим массивом (ArrayList) в Java:

1. **Изменяемость размера**. Массивы имеют фиксированный размер, ArrayList может изменяться. 3
2. **Типы данных**. Массивы могут хранить примитивы, ArrayList — только объекты. 3
3. **Методы управления**. ArrayList предоставляет множество методов (add(), remove(), get()), массивы — нет. 3

13. Вложенные, локальные и анонимные классы.

1. **Вложенные классы** — это классы, определённые внутри другого класса. 13 Область действия вложенного класса ограничена областью действия внешнего класса. Вложенный класс имеет доступ к членам (в том числе закрытым) того класса, в который он объявлен. 1
2. **Локальные классы** — объявленные внутри блока кода и не являющиеся членом обрамляющего класса. В этом случае можно рассматривать класс как локальную переменную типа класс. 3 Основная характеристика локального класса — возможность декларирования в любом блоке кода, допускающем объявление переменных. 5
3. **Анонимные классы** — наследуемые от какого-либо класса, классы, в которых при объявлении не задано имя класса. 3 Используются тогда, когда нужно переопределить метод класса или интерфейса. Класс одновременно объявляется и инициализируется. Могут быть объявлены не только в методе, но и внутри аргумента метода.