Abstractions

Хороший способ забыть о целом - пристально рассмотреть детали.

Чак Паланик

Конкретные типы

Помните первый пример с телегой из прошлой презентации на тему «полиморфизм»? В базовом классе Transport был метод drive(), а остальные классы-наследники переопределяли поведение этого метода. Car, Bike и Telega из того примера – это **конкретные, завершённые типы** данных для объектов, которые реально будут использоваться в приложении, и получат своё физическое воплощение, например, в виде 2D или 3D-моделей, с определёнными текстурами, и может, ими даже кто-то будет управлять, как в GTA 5.

Абстрактные типы

Но вот что касается типа Transport, то в приложении создавать объекты такого типа скорее всего не придётся никогда. Этот класс был создан исключительно для того, чтобы **задать общие принципы** дальнейшей работы всех транспортных средств-потомков. Объекты именно с типом Transport создавать бессмысленно, так как они не получат конкретного физического воплощения в виде модели с текстурами или даже просто двумерной картинки. Более того, для того, чтобы избежать логических ошибок в клиентском коде, делать это крайне нежелательно.

Абстрактное животное

Что будет, если попросить дизайнера нарисовать **животное**?

Примеры абстрактных типов

Похожая картина наблюдается в отношениях классов **Figure** – Circle, **Unit** – **Dragon** – RedDragon, **SuperHero** – Batman, **Fruit** – Apple, **Animal** – **Cat** – BritishCat, **Collection** – ArrayList, **Weapon** – Shotgun, **Device** – MobilePhone, и тд. Как небольшое исключение из этой закономерности можно привести классы Student – Aspirant, Passport – ForeignPassport, Object.

Получается, что конкретными можно назвать лишь те типы данных, которые обычно находятся на самом нижнем уровне в иерархии наследования.

Ключевое слово abstract

Для того, чтобы у программиста-пользователя в его клиентском коде не было возможности создать объект абстрактного типа, базовые классы помечают как abstract:

```
abstract class Animal { … }
Animal a = new Animal(); // ERROR!
```

Абстрактный vs. конкретный

Каких-либо отличий между конкретным и абстрактным классом нет. Абстр. классы поддерживают работу с полями, конструкторами, обычно содержат реализацию методов – просто **нельзя создать объект абстрактного типа**. Однако, очень важно учесть, что ссылки абстрактного типа создавать можно и даже нужно ☺

```
Animal a = new BritishCat(); // OK!
```

Конкретные реализации методов

Вернёмся к примеру с классом Transport. Нужна ли в этом классе конкретная реализация метода drive()? Нет, не нужна, поскольку не понятно, каким именно образом должно будет ехать некое **абстрактное** транспортное средство. Класс Transport важен для проектирования исключительно как абстрактный базовый класс, определяющий общие принципы работы его потомков.

Абстрактные методы

В языке Java существует возможность зафиксировать такого рода незавершённость, абстрактность метода на уровне языка. Для этого метод помечается словом abstract. Абстрактный метод – это метод, который в базовом классе только объявляется, но не определяется (не реализуется). Например, метод drive() как раз таки можно сделать абстрактным в классе Transport, потому что ничего конкретного о том, как перемещается некое неопределённое транспортное средство, мы не знаем.

Синтаксис объявления

```
abstract class Transport {
    double speed;
    abstract void drive();
}
```

Обратите внимание, что **фигурных скобок и тела у абстрактного метода нет**! Класс, в котором есть хотя бы один абстрактный метод, также должен быть помечен как абстрактный. Впрочем, наличие абстрактных методов в АК не является обязательным - в составе АК могут быть и обычные неабстрактные методы с реализациями, и это лишь ускорит процесс разработки новых типов!

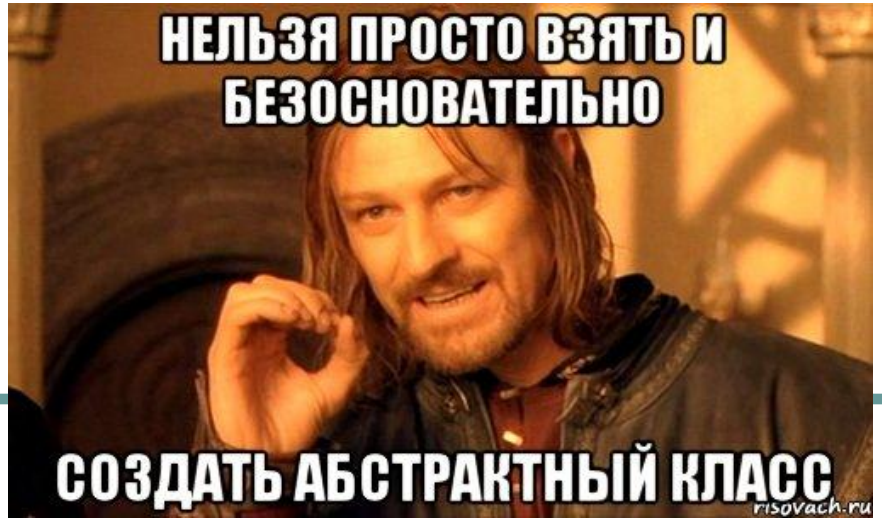
Запрет создания объектов

Если вдруг появится непреодолимое желание создать объект абстрактного типа, то придётся сначала объявить класс-потомок этого абстрактного класса, переопределить (реализовать) в классе-потомке все абстрактные методы, и лишь затем создать объект, но – всё-таки производного типа.

И кстати, не бывает абстрактных конструкторов и абстрактных статических методов (угадайте, почему? ☺)

АК – это заготовка

Абстрактный класс – это ЗАГОТОВКА для будущих классов. Нет смысла делать класс абстрактным, если не планируется наследование от него (попробуйте сделать АК final ☺) Наличие абстрактных методов имеет большое значение, так как наряду с обычными методами, они также задают общее поведение для всех будущих подклассов.



Пример кода

https://git.io/vrNeS

Практика

- Применить ключевое слово abstract в заданиях **MyCollection**, **Figure**, **Weapon**, **Device**.
- Убедиться в невозможности создания объектов базовых типов.
- Сделать абстрактные методы в суперклассах, и переопределить их в классах-наследниках.

Нужно больше абстракции

В языке Java существует возможность выйти на ещё более высокий уровень абстракции, когда для типа в принципе запрещено определять какие-либо свойства (поля). Интерфейс – это особая синтаксическая конструкция в коде программы, позволяющая описать некий абстрактный список услуг (действий).

Интерфейс - набор действий

И если класс (в том числе абстрактный) больше определяет саму сущность, включает описание характеристик этой сущности, т.е. отвечает на вопрос «**кто я** / что я», то интерфейс определяет только одно действие или же набор действий, выражаемых глаголами, т.е. отвечает на вопрос "**что я могу делать**".

Понятие реализации

Когда мы переносим в программный код сущность из реального мира, мы определяем наиболее значимые её характеристики (формируем список полей). Это – наша **РЕАЛИЗАЦИЯ** сущности, то, **ЧЕМ** конкретно она будет представлена в программе.

Интерфейс – это поведение

В реальности, свойства любой сущности (например, студента) включают также **действия**, которые она может совершать сама, или же которые можно совершать над ней. Например, студенту можно назначить старосту, поднять его рейтинг, студент может учиться, отдыхать, работать и тд. Чтобы задать **ПОВЕДЕНИЕ** (**интерфейс**) студента, мы используем методы.

Описание поведения

Интерфейсы описывают только лишь поведение сущности (что может делать объект, реализующий интерфейс, через свои методы - какие он может генерировать события, и каким способом он может рассказывать о себе). **Классы же описывают как своё поведение, так и внутреннее состояние** (чему равно в каждый отрезок времени каждое из полей класса).

Несколько примеров

Например, интерфейс **Comparable** говорит лишь о том, что объекты, его реализующие, можно как-то сравнивать. Он ничего не говорит о сущности самих объектов, об их типах, или о том, как будет проходить сравнение. Наличие интерфейса **Cloneable** гарантирует, что некий объект можно скопировать. **Iterable** – намекает на то, что элементы коллекции можно перебрать циклом foreach, и т.д.

Интерфейс vs. реализация

Главное отличие класса от интерфейса — в том, что **класс состоит из интерфейса и реализации**. Любой класс всегда неявно объявляет свой интерфейс — то, что доступно при использовании класса извне (набор публичных методов). В то же время, в состав класса могут входить приватные поля – а это уже часть реализации.

Определение

Интерфейс - это совокупность методов и правил взаимодействия элементов системы между собой.

- Интерфейс двери - наличие ручки
- Интерфейс автомобиля - наличие руля, педалей, рычага коробки передач
- Интерфейс дискового телефона - трубка + дисковый набиратель номера

Когда мы используем эти объекты, мы уверены в том, что сможем использовать их определённым образом. Это происходит благодаря тому, что мы знакомы с их интерфейсом.

Ручки в дверях метро ☺

Графический интерфейс

Почему графическую часть программы часто называют интерфейсом? Потому что она определяет, каким образом мы сможем использовать основную функциональность, заложенную в программу. То есть, **интерфейс определяет, каким образом мы можем использовать тот или иной объект**.

Ещё пример

Предположим, существует класс Table (стол), у которого есть методы работы с ним: поесть сидя за ним, поставить, собрать, продать, отпилить ножки и прочее. Из класса можно попытаться выделить интерфейс Мебель. Понятно, что не на всякой мебели можно поесть или отпилить ножку, но зато любую мебель можно поставить и собрать. И если вдруг появится класс Chair (стул), который также реализует интерфейс Мебель, то со стулом и со столом можно будет обращаться одинаково - в этом и состоит **смысл интерфейсов - в унификации работы с разнотипными объектами**.

Взаимодействие элементов

Интерфейсы являются основой взаимодействия всех современных информационных систем. Если интерфейс какого-либо объекта (персонального компьютера, программы, функции) не изменяется (стабилен, стандартизирован), это даёт возможность модифицировать сам объект, не перестраивая принципы его взаимодействия с другими объектами (так, например, научившись работать с одной программой под Windows, пользователь с лёгкостью освоит и другие — потому, что они имеют однотипные элементы интерфейса).

Простыми словами

По сути, интерфейс – это тот же абстрактный класс (нельзя создать объект интерфейсного типа), но в интерфейсах не может быть полей, конструкторов, и все методы – публичные, нестатические и абстрактные. Создаётся в коде с помощью специального ключевого слова interface.

Интерфейс – это контракт

Интерфейс содержит описание (декларацию) методов, реализацию которых обязуется осуществить класс-наследник. То есть, как и абстрактные классы, интерфейсы создаются не просто так, а для того, чтобы их в последствии кто-то реализовал. **Это своего рода "контракт" (протокол взаимодействия) для всех классов, желающих "имплементировать" требования интерфейса.** Любой класс может реализовать любое количество интерфейсов. Но при этом класс должен реализовать все их методы!

Абстрактные классы vs.интерфейсы

- Абстрактные классы используются только тогда, когда есть тип отношений «**is a**», в то время как интерфейсы могут быть реализованы классами, которые вообще никак не связаны друг с другом (задействуется другой тип отношений – **implements**).

- В Java класс может реализовать несколько интерфейсов, а отнаследоваться можно только от одного абстрактного класса.

Пример кода

https://git.io/vrNfB

Ещё пример (openable)

https://git.io/vrNUm

И ещё пример (superheroes)

https://git.io/vrNTa

Shared constants

https://git.io/vrNkL

```
run:
Я ль на свете всех милее?
YES
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 9 секунды)
```

Static member class

https://git.io/vrNlg

Реализация методов интерф.

https://git.io/vokD4

http://stackoverflow.com/questions/23721759/functionalinterface-comparator-has-2-abstract-methods

Реализация Cloneable

https://git.io/vrNYf

Реализация Comparable

https://git.io/vrNOi

Реализация Comparator

https://git.io/vrN3A

Iterator and Iterable

https://git.io/vrNZ0