- RTTI - механизм динамического определения типов, позволяет получать и использовать информацию о типах во время выполнения программы
- Необходимость в RTTI
- На примере семейства геом фигур
- Стремление к манипулированию приемущественно базой (метод drow можно вызвать через базу)

Добавляем элемент в контейнер(где все обджект), извлекаем обратно - динамически определяется тип фигуры.

Благодаря RTTI можно запросить точный тип объекта, на который указывает ссылка базового типа Shape

Объект Class

Хранит сведения о классе, существует для каждого класса

Чтобы создать объект вызывается загрузчик класса. Существует один первичный загрузчик классов - часть реализации JVM, загружает доверенные классы

Классы загружаются при их первом использовании -> динамическая загрузка

Способ получить ссылку на объект class - вызвать метод forname(string) определенного класса. Можно вызвать object:getClass(). Можно определить является ли интерфейсом, вывести простое\каноническое имя

- Синтаксис приведения типа
- Метод cast
- Получает аргумент-объект и преобразует его к типу ссылки на Class/
- Полезно, если нельзя использовать обычное приведение типа
- House h = houseType.cast(b) или h = (House)b
- RTTI -> классическое преобразование типов (в скобочках), объект класс, представляющий тип вашего объекта
- Существует третья форма -
- If (z instanceof Dog){((Dogx).bark();)}
- Предложение if проверяет принадлежит ли x DOG? Если да то проведение нисходящего преобразования
- Обычно - поиск одного определенного типа
- В методе countPets - массив заполняется объектами Pet с использованием PetCreator, затем каждый подсчитывается с iinstanceof
- Использование литералов class
- Контейнер заполняется всеми типами Pet. Еще один список, хранящий все все типы, созданные случайным вызовом. Проверяется сначала присутсвие в первом контейнере, а потом добавление во второй

- У объекта Class можно запросить информацию о базе, можно использовать newInstance как виртуальный конструктор
- Литералы class - Класс.class + .TYPE (для оберток)
- 1Загрузка - находит байткод и создает на его основе объект Class
- 2. Компоновка - проверяется байт-код класса, выделяется помять для статических полей, разрешаются ссылки на используемые классы
- 3. Инициализация. Статич. Иниц+блоки статич иниц + иниц суперкласса если есть
- Инициализация откладывается на столько, на сколько возможно
- Если статическое неизменное значение - константа времени компиляции
- Сслыку на класс можно связать с любым объектом класс, параметризованную только с объявленным типом.
- Для ослабления ограничений - метасимвол ? - Class<?> intClass = int.Class, а потом можно - intClass = double.class;
- Чтобы создать ссылку на class, ограниченную типом\подтипами надо - <?extends ClassName>
- Параметризация была добавлена для того, чтоб можно было раньше узнать об ошибках
- С обычными ссылками на класс о них узнаете только после выполнения.

- Динамическая проверка типа
- Class.isInstance() - динамич, в аргументах класс
- Метод isInstance() избавил от необходимости использовать цепочки instanceof
- Class.isAssignableFrom(superClass) - проверяет принадлежность переданного объекта нужной иерархии

- Зарегестрированные фабрики
- Использование фабричного метода и класса фабрики для создания объекта

Instanceof и сравнение объектов Class

Важно различать instance от сравнения самих классов

Instanceof и isInstance - одинаков. Результаты == и equals тоже совпадают

Динамическая информация о классе

Компилятор должен располагать информацией о классе с которым вы работаете

Может не располгатать если - считываете байт-код, который на самом деле класс, при работе на удаленных платформах. Класс вводит понятие отражение, для отражение с классами Field, Method, Constructor. Объекты объявляются для создание соответствующих членов неизвесного класса. Информация о неизвестном объекте доступна во время выполнения программы

- Механизм отражение - JVM видит неизвестный объект и видит, что этоо класс, но файл .class не доступен -> но методы можно вызвать
- Извлечение информации о методах класса
- Отражение существует для возможности сереализации, для компонентов JavaBeans. Но иногда информация о классе очень нужна
- Программа, выводящая список методов класса. Напрямую видны метды реализованные в классе, но может быть еще методы, доступные из базы. Найти - сложно и долго
- GetMethods() - помогает найти методы

- Динамические заместители - proxy - один из основных паттернов проектирования. Представляет собой объект, который подставляется на место "настоящего объекта "
- Для предоставления дополнительных или других операций. - подразумевает взаимодействие с "настоящим" объектом, поэтому заместитель - функции посредника
- Proxy - реализует интерфейс заменяегого класса, вызывает его функции(сам класс в аргументы)
- Заместитель нужен при желании включить дополнительные операции в любом месте, отличным от настоящего объекта.

- Динамические заместители - объект заместителя создается динамически, вызовы методов тоже динамически. Все вызовы к обраб, которы решает, что делать. Методу Proxy.newProxe.Instance передается загрузчик классов (getClassLoader), интерфесы класса и обработчик вызовов (реализация интерфеса).

Проект[1]. Напишите систему, использующую динамические заместители для реализации транзакций: заместитель закрепляет транзакцию, если опосредованный вызов выполнен успешно (т. е. не возбудил исключений) или выполняет отмену в случае неудачи. Закрепление и отмена должны работать для внешних текстовых файлов, что выходит за границы исключений Java. Уделите особое внимание атомарности операций.

Null объекты
Появляется, если значение по умолчанию объекта null и он не объявлен
Самого объекта null не существует, но иногда можно попробывать использовать. Например, если информации об объекте не достаточно, можно использовать нул-объект в качестве соответствующего поля.(объект имитирующий поведение нулевой ссылки)
Но чаще всего достаточно проверять ссылки на null

- Фиктивные объекты и заглушки (вариация null объектов)
- Оба объекта - дублеры настоящего
- Фиктивные объекты - легковесны, самотестируемы, может быть несколько
- Заглушки - вовзращают представляемые данные, больше весят, повторно используются между тестами
- Заглушка - сложный объект, реализующий множество функций (например класс база имеет функцию, необходимую только для одного наследника. Для обеспечения полиморфности определяется именно в базе, тогда один ребенок должен будет содержать этот метод, а другие использовать заглушку(но можно просто обозначить метод, подходящий всем))
- Фиктивные объекты - для выполнения разнообразных функций(создание множества объектов)
- Интерфейсы и информация типов
- Интерфейс не обеспечивает стопроцентной гарантии снижения связности (можно добраться к фактическому типу с помозью приведения, некоторые доп функции, не указанные в нтерфейсе не будут работать) чтоб этого не было можно скрыть интерфейс на уровне пакета, оставив только одну функцию, возвращающую класс
- Но callHiddenMethod - позволяет увидеть даже закрытые методы
- SetAccesible(true)- позволяет получить доступ к классу