Повторное использование классов

- Композиция - механизм построение нового класса на основе существующих
- Наследование - представление нового класса как специализацией существующего

- Композиция
- В случае включения в создания класса объекта другого класса, идет объявление его ссылки. Инициализировать ссылку можно в следующих местах: в точке определение объекта, в конструкторе, перед использованием (отложенная инициализация), с использованием инициализации экземпляров
- Наследование
- Class Children extends Parent
- //метод main всегда является открытым и доступным
- В случае пакетного доступа, если ребенок не в пакете, то доступ только к публичным членам класса
- Все поля private, методы public + protected. Наследник получает методы родителя в интерфейсе. Для использование тех же функций в наследнике - (super)
- При создании объекта производного класса создается подобъект базового класса-> в объекте производного упакован базовый

- Важна правильная инициализация базового класса -> инициализация происходит в конструкторе. При создании ребенка, конструируется наследник, а потом ребенок.
- В случае с конструкторами аргументами, если родитель не имеет конструктора по умолчанию, вызов конструктора оформляется явно с указанием ключевого слова super
- Если не сделать этого, то компилятор пожалуется + конструктор базы дожен быть первым методом
- Делегирование (агрегация)
- Экземпляр существующего класса включается в класс + его методы используются
- Компилятор следит за конструкторам базового класса, но не следит а конструкторами внедряемого.
- Обеспечение правильного завершения
- Уборщик мусора хорошо, но ручная уборка нужна в случае операций, требующих завершающее действие (нарисовать что-то и стереть за собой)
- Сокрытие имен
- Если какой-то из методов в базе перегружен (одно имя с разными аргументами), переопределение не скроет базовые версии -> можно использовать с разными аргументами, функция выбирается в зависимости от них. В таком случае оверайд не уместно

- PROTECTED
- Используется, когда необходимо спрятать от всех, но не от наследников (можно исп. При создании объекта или при специфических методах) + доступ в пределах пакета.
- Восходящее преобразование типов
- Новый класс является разновидностью существующего - описание наследования. Любое сообщение, которое можно отправить родителю доступно ребенку. Преобразование от производного типа к базовому требует движение вверх - поэтому восходящее. Производный - надстройка базового, обязан включать все методы базового - > при переходе от производного к базовому возможна утечка методов, но не приобретение
- Критерий выбора между композицией и наследованием - собираетесь ли вы использовать восх преобр?
- Ключевое слово final
- Это нельзя изменить (по причине соблюдения архитектуры/эффективности)
- Неизменяемые данные
- Поезны - константа времени компиляции, значение инициализируемое во время работы программы
- Static и final поле существует в памяти в единственном экземпляре и не может быть изменено (записывается заглавными буквами, слова с подчеркиванием)

- Final со ссылками на объекты - делает постоянной ссылку(у примитивов - значение) После связи ссылки с объектом, она не сможет указывать а другой объект, при этом сам объект может изменяться (например можно увеличить)
- Пустые константы
- Поля, объявленные финальными, но без начального значения. Значение надо присвоить перед использованием
- Неизменные аргументы
- Ключевое слово final в списке аргументов - метод не может изменить значение, на которое указывает передаваемая ссылка. Внутри метода их нельзя менять, можно использовать их значения. (используется для анонимных внутренних классов)
- Неизменные методы
- Используются для блокировки (производные классы не могут изменить значение) - поведение метода не изменится при наследовании. Любой приватные метод класса - косвенно неизменный. Final можно добавить к закрытому методу, но его присутствие не повлияет. При попытке переопределения закрытого метода просто объявляется новый метод с тем же именем
- Неизменные класса
- Объявляя класс неизменным вы показываете, что не собираетесь использовать класс в качестве базового и запрещаете делать это другим - структура вашего класса - постоянна. Поля могут быть изм\не изм. Final = запрет на наследование

- Определяя метод финальным надо учесть возможность повторного использование
- Инициализация и загрузка классов
- Сначала загрузка при запуске, далее инициализация, затем программа начинает работу. Точка первого использования = точка загрузки кода и иниц стат переменных. При загрузке производных классов автоматически происходит загрузка базового, потом статик базы потом статик производного
- При создании объекта - сначала полям значения по умолчанию, потом конструктор базы, потом иниц перем, потом тело конструктора

- Наследование и композиция позволяют создавать новые типы на основе существующих. Композиция для повторного использования реализации, наследование - повторное использование интерфейса. Композиция предпочтительнее при начальном проектировании