* **Анонимни класове** - Вид **вътрешни класове**, но **без имена**. Анонимните класове **не могат да дефинират в своето тяло конструктори** и за инициализация използват конструкторите на предшественика си.
* **Идентификация по време на изпълнения** (Run-time Type Identification - **RTTI**) - Идентификацията на типове по време на изпълнение на програмата е мощен механизъм, който позволява да разберем типа на дадена инстанция, имайки само референцията към нейния базов клас.
* Оператор **instanceof**
* **RTTI техниката се базира на специализирания обект от тип Class, в който Java съхранява метаинформация за съответния тип клас**. Когато извикаме оператора "instanceof", той използва метаданните, представяни от посочения от нас обект и клас, за да се провери израза.
* **instanceof**vs. **Class.isAssignableFrom()** - Използването на instanceof е по-добър и по-елегантен подход.
* **Преобразуване на един референтен тип към друг референтен тип (кастване).**
* **Композиция**- Създаването на обект в контекста на даден друг клас (добавяне на полета от референтен тип). Употребата на композиция е честа, например, винаги когато добавяме полета от тип String, Integer, ArrayList и др. Дава лесен начин за преизползване на код.
* **Композиция**vs. **Наследяване** - за повече информация отворете прикачената снимка.
* **Enumeration**(Изброен тип) - Конструкция, наподобяваща клас, в която можем да декларираме само логически свързани константи.
* **Вътрешни класове**- Езикът Java позволява създаването на дефиниции на класове вътре в други класове. Позволява се логическото им групиране и контролиране на видимостта. Вътрешните класове могат да **достъпват полета и методи от обгръщащия ги клас**, Вътрешните класове **не могат да съдържат статични полета, класове или методи**.
* **Статични вложени класове** - Не е необходимо съдържащият клас на вътрешни класове да бъде инстанциран, за да се използват. Имат директен достъп до статичните полета и методи на съдържателя, а ако разполагат с негова инстанция могат да използват и нестатичните му ресурси. Не притежават неявна референция към външния клас, който ги обгръща.
* **Абстрактен клас** - Не може да се създават обекти от тип абстрактен клас. Дефинира се с ключовата дума "abstract". Всеки клас, който има поне един абстрактен метод, трябва да бъде деклариран като абстрактен. Може да се дефинира абстрактен клас без той да съдържа абстрактни методи вътре в себе си. Абстрактните класове могат да съдържат в себе си, както абстрактни, така и други методи.
* **Абстрактен метод** - Mетод без тяло, само с декларация. Дефинира се с ключовата дума "abstract".
* **Чист абстрактен клас** - Съдържа в себе си само абстрактни методи. Не може да съдържа член-променливи.
* **Виртуален метод** - Метод, който може да бъде пренаписан (override). По подразбиране всички  методи в Java са виртуални. Ако не искаме да бъдат такива ги дефинираме с ключова дума **final**. Тогава никой клас-наследник няма да може да пренапише този метод.
* **Интерфейс**- Дефинира поведение без изрична имплементация. Един клас може да имплементира много интерфейси (използва се ключова дума "implements"). В интерфейсите може да има само абстрактни методи и константи. Дефинира се с ключовата дума "interface". Класът, който имплементира интерфейс трябва да имплементира всички методи от интерфейса. Наследници на този абстрактен клас. Най-добрият начин да се реализира абстракция е да се работи с интерфейси.
* **Разлики между интерфейс и абстрактен клас**
* Списъчни структури от данни в Java (**ArrayList**, **LinkedList**, и др.)
* Класът **ArrayList** - Динамично разширяем масив. Важни методи: get(), add(), remove(), clear().
* **Принципи на ООП** (абстракция, капсулация, наследяване, полиморфизъм);
* **Наследяване**на класове: Не е възможно множествено наследяване с класове в Java (един клас може да има само един клас-родител - single inheritance); Йерархия от класове и преизползване на код;
* **Капсулация:**скриване на детайлите на класовете, посредством модификатори за достъп;
* **Модификатори за достъп**: public, private, protected, package, default.
* **Абстракция:**игнорираме детайлите и виждаме само това което ни интересува**;**
* **Полиморфизъм:**дефинира как се работи по еднакъв начин с различни обекти, които дефинират специфична имплементация на абстрактно поведение (override и overload)**;**
* **Пренаписване**на методи (*Override*): Отбелязва се с анотация @Override преди дефиницията на метода. Наследените от базовия (родителски) клас методи могат да се пренаписват. Това означава да се подмени имплементацията, като оригиналният код от базовия клас се игнорира за сметка на пренаписания нов код. В това се изразява, така нареченият, полиморфизъм.
* Метод **toString()** и override от базовия клас Object;
* **Варианти**на методи (*Overload*): когато декларираме един метод, чието име съвпада с името на друг метод, но сигнатурите на двата метода са различни по списъка на параметрите (броят или подредбата им), казваме че имаме различни **варианти** на този метод (overload); Пример за overload са **конструкторите**на класовете.
* Разлики между **клас**и **обект**.
* **Статични**(static) и **нестатични**(non-static) членове на клас.
* **Статичните**(static) полета и методи са свързани с класовете, а не с конкретен обект от даден клас. Използват се без да има създадена инстанция/обект  на класа. Статично поле или метод се дефинира с ключовата дума static (public static String helloWorld = "Hello World!";).
* **Нестатичните**(non-static) полета и методи са свързани с обект от даден клас. Следователно, нестатичните членове на клас се извикват с помощта на създаването на обект. По подразбиране всички променливи и методи са non-static.
* **Пакети**(packages) в Java - пакетите позволяват на разработчиците да групират своите класове на база определени критерии.