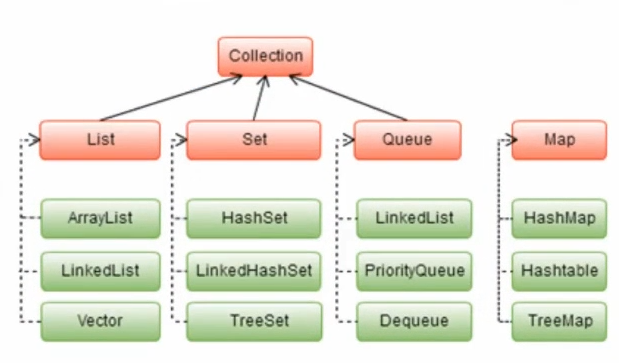
1.**Drzewko dziedziczenia Collections**



**2.Czy operator % może zwrócić liczbę zmiennoprzecinkową?** Tak, jeśli robimy na double.

**3. co oznacza … w metodzie?** (czy to jest varargs?)

Przyjęcie dowolnej liczby elementów tego samego typu

**4. Czy switch operuje na warunku logicznym?** W kodzie nie.

5. **Zmienne globalne inicjalizowane są wartościami domyślnymi bo są składowymi klasy. A jeśli tworzony jest jakiś obiekt to JVM musi wiedzieć ile miejsca zainicjalizować na stosie. Zmienne lokalne nie są domyślnie inicjalizowane bo leżą na stercie.**

**6. Wzorzec projektowy** – wskazówka w jaki sposób rozwiązać jakiś problem

**7. SOLID – dobre praktyki programowania**

|  |  |  |
| --- | --- | --- |
| **S** | SRP[[2]](https://pl.wikipedia.org/wiki/SOLID_(programowanie_obiektowe)#cite_note-SRP-2) | **Single responsibility principle (**[**Zasada jednej odpowiedzialności**](https://pl.wikipedia.org/wiki/Zasada_jednej_odpowiedzialno%C5%9Bci)**)**  [Klasa](https://pl.wikipedia.org/wiki/Klasa_(programowanie_obiektowe)) powinna mieć tylko jedną odpowiedzialność (nigdy nie powinien istnieć więcej niż jeden powód do modyfikacji klasy). |
| **O** | OCP[[3]](https://pl.wikipedia.org/wiki/SOLID_(programowanie_obiektowe)#cite_note-OCP-3) | **Open/closed principle (**[**Zasada otwarte-zamknięte**](https://pl.wikipedia.org/wiki/Zasada_otwarte-zamkni%C4%99te)**)**  Klasy (encje) powinny być otwarte na rozszerzenia i zamknięte na modyfikacje. |
| **L** | LSP[[4]](https://pl.wikipedia.org/wiki/SOLID_(programowanie_obiektowe)#cite_note-LSP-4) | **Liskov substitution principle (**[**Zasada podstawienia Liskov**](https://pl.wikipedia.org/wiki/Zasada_podstawienia_Liskov)**)**  Funkcje które używają wskaźników lub referencji do klas bazowych, muszą być w stanie używać również obiektów klas dziedziczących po klasach bazowych, bez dokładnej znajomości tych obiektów. |
| **I** | ISP[[5]](https://pl.wikipedia.org/wiki/SOLID_(programowanie_obiektowe)#cite_note-ISP-5) | **Interface segregation principle ([Zasada segregacji interfejsów](https://pl.wikipedia.org/w/index.php?title=Zasada_segregacji_interfejs%C3%B3w&action=edit&redlink=1" \o "Zasada segregacji interfejsów (strona nie istnieje)))**  Wiele dedykowanych interfejsów jest lepsze niż jeden ogólny. |
| **D** | DIP[[6]](https://pl.wikipedia.org/wiki/SOLID_(programowanie_obiektowe)#cite_note-DIP-6) | **Dependency inversion principle (**[**Zasada odwrócenia zależności**](https://pl.wikipedia.org/w/index.php?title=Zasada_odwr%C3%B3cenia_zale%C5%BCno%C5%9Bci&action=edit&redlink=1)**)**  Wysokopoziomowe moduły nie powinny zależeć od modułów niskopoziomowych - zależności między nimi powinny wynikać z [abstrakcji](https://pl.wikipedia.org/wiki/Abstrakcja_(programowanie)). |

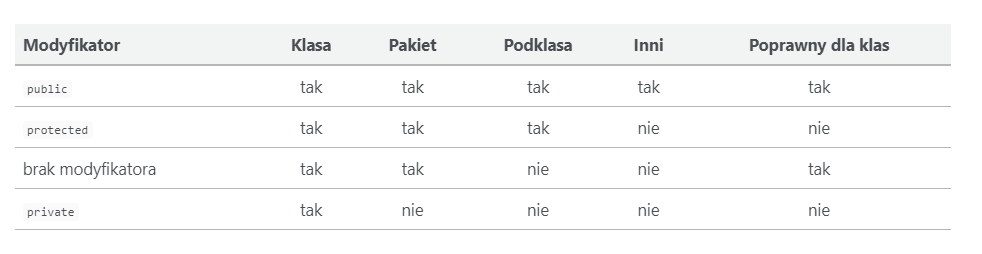
**8. Przeciążanie metod**: wystarczy tylko taka sama NAZWA metody

**9.Czy aby przekazać obiekt, potrzebujemy uchwytu?** Nie, można przecież użyć new StringBuilder()

**10. Czy można rzutować booleana na inta?** NIE !!! W Javie nie.

**11.JVM, tłumaczy kod bajtowy na binarny, zarządza pamiecią i optymalizuje**

12. Brak modyfikatora oznacza package private (widoczność w katalogu) – to domyśla widoczność w Javie



**13. Elementy tablicy nie muszą być znane przed implementacją. Musi być znana ich liczba i typ.**

**14.Od Java 11 możemy bawić się GarbageCollectorem**

**15.Czy dwa obiekty mogą zwrócić ten sam hash code?** To klucz obliczony na podstawie wartości zmiennych. Dlatego może ta wartość być taka sama. Dlatego jeśli dwa hashe są takie same to należy się jeszcze upewnić equals() ze dwa obiekty są takie same.

**16.finalize().** Zanim GC usunie jakiś obiekt to wywoła tę metodę obiektu

**17.finally** to w ca tchu

**18.final,** stała

**19. adnotacja** – dodatkowe info, meta dane

**20.maven** – budowniczy projektu

**21. interfejs** – cos jest czymś

**22. Klasa abstrakcyjna** – nie da się utworzyć jej obiektów

**23. kontroler w mvc** – odczytuje zapytania

24.@AutoWired – wstrzyknięcie zależności,

So, to answer your question, @Autowired is Spring's own (legacy) annotation. @Inject is part of a new Java technology called CDI that defines a standard for dependency injection similar to Spring. In a Spring application, the two annotations works the same way as Spring has decided to support some JSR-299 annotations in addition to their own.

Better use @Inject all the time. Because it is java configuration approach(provided by sun) which makes our application agnostic to the framework. So if you spring also your classes will work.

If you use @Autowired it will works only with spring because @Autowired is spring provided annotation.

25. @Service, @Component, @Entity - wszystkie zamieniają w ziarno

Spring 2.5 introduces further stereotype annotations: @Component, @Service, and @Controller. @Component is a generic stereotype for any Spring-managed component. @Repository, @Service, and @Controller are specializations of @Component for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.

│ @Component │ generic stereotype for any Spring-managed component │

│ @Repository│ stereotype for persistence layer │

│ @Service │ stereotype for service layer │

│ @Controller│ stereotype for presentation layer (spring-mvc)

Jedna funkcjonalność, zbiór serwisów, encja

**26. shutdown hook**- obszar z którego wykonywane są polecenie zanim JVM zakończy działanie programu

