|  |  |
| --- | --- |
| Co to jest String Pool | String Pool jest to miejsce w Heapie gdzie przechowywane są stworzone Stringi. Jeżeli String tworzony jest przez `new` to nie jest wrzucany do Poola  String s1 = "Cat";  String s2 = "Cat";  String s3 = new String("Cat");  s1 == s2; //true bo wskazują na to samo miejsce w String Pool  s1 == s3; //false bo wskazuja na inne miejsca w pamięci  Istnieje metoda `intern()`, która przerzuci String do String Pool. |
| Róznica między String, StringBuffer a StringBuilder | * String: immutable, używa StringBuffera do konkatynacji * StringBuffer: przez to że sam String jest immutable to Buffer służy do konstrukcji, konkatynacuji stringów, substringowania itd * StringBuilder: nie-threadsafe wersja StringBuffera - szybsza ale tylko   przy aplikacjach jednowątkowych (kompilator domyślnie używa go przy "+")  StringBuilder i StringBuffer wewnętrznie przechowuja tablicę charów. |
| Co to jest i jak działa GC (Garbage Collector) | Garbage Collector jest to Daemon thread działający cały czas w tle JVM, który na celu ma dekstrukcję niepotrzebnych obiektów i zwalnianie miejsca w Heapie.  GC usuwa elementy, które są "unreachable". Możemy sprawić że obiekt jest unreachable poprzez:   * znullowanie referencji do niego * nadpisaniem referencji do niego * jeżeli obiekt tworzony jest wewnątrz metody * Island of Isolation (kiedy dwa obiekty wskazują jedynie na siebie, bez referencji zewnętrznych)   Przed usunięciem obiektu GC wywołuje na nim metodę `finalize()`.  GC jest wywoływany wtedy kiedy JVM zdecyduje że trzeba zwolnić pamięć na Heapie  Metody `System.gc()` oraz `Runtime.gc()` wysyłają zapytanie do GC, ale nie dają 100% pewności, że GC zadziała w tym momencie. |
| Czym jest JVM | Jest to wirtualna maszyna która pozwala komputerowi na odpalenie programów, które zostały skompilowane do Java bytecodu.  Dzięki konkretnym specyfikacjom, które musi spełniać każda JVM autorzy programów mogą mieć pewność że ich program będzie działał tak samo na większości urządzeń. |
| Różnica między JDK, JRE, JVM | * JVM opisano na górze * JRE - Java Runtime Enviroment - środowisko do odpalnia programów Java * JDK - Java Development Kit - oprócz środowiska do odplania programów posiada również narzędzia pozwalająca na ich tworzenie |
| Czy Java jest w 100% obiektowa | Nie.  Posiada typy proste:   * int * double * long * float * boolean * char * byte |
| Czym jest konstruktor | * Konstruktor jest metodą, którą musi posiadać każda klasa i służy do tworzenia nowych obiektów na podstawi tej klasy używając słowa kluczowego `new` * Każda klasa ma w sobie domyślny konstruktor, który ustawia polom klasy jej domyślne wartości. * Konstruktor jako metoda nic nie zwraca i musi się nazywać tak jak klasa którą tworzy. |
| Różnica między equals() a ==? | equals()  prównuje wartości obiektów używając wbudowanych w ich klasy metod `equals()`  ==  porównuje referecje do miejsc w pamięci, czy porównywane zmienne wskazują na ten sam obszar pamięci |
| Przeciążanie (overloading) i przesłanianie (overriding) | Overloading metody polega na stworzeniu nowej metody o takiej samej nazwie, ale innej sygnaturze - inne parametry, inne ciało, inny typ zwracany  Overriding - pojawia się przy dziedziczeniu albo implementacji interfejsów polega na "nadpisaniu" dziedziczonej lub implementowanej metody nowym ciałem.  Parametry i typ zwracany metody pozostają bez zmian (typ zwracany może być podtypem tego co zwraca overridowana metoda |
|  |  |

10. Słowo kluczowe `volatile`

Jeżeli mamy przypadek kiedy jeden wątek zmienia zmienną, a inny ją odczytuje

wątek odczytujący może wziąć wartość zmiennej zanim wątek nadpisujący zdąży przenieść

jej zaktualizowaną wartość z pamięci cache CPU do głównej pamięci programu.

Słowo `volatile` zapewnia że zmianny wartośći będą od razu zapisanywane do głównej pamięci,

i odczyt zmiennej będzie zawsze się pokrywał z jej aktualną wartością.

11. sleep() oraz wait()

`sleep()` używa się aby wstrzymać działanie wątku na określoną ilość czasu

`wait()` wstrzymuje działanie wątku aż go wybudzimy `notify()` lub `notifyAll()`

Ważną różnicą jest to że `wait()` zwalnia lock trzymany przez wątek, a `sleep()` nie.

`wait()` użyjemy w aplikacji wielowątkowej do komunikacji między wątkami, `sleep()`

po prostu do wstrzymania działania aplikacji na jakić czas.

12. Reflection w Javie

Reflection polega na zdolności programu do "introspekcji" czyli dostępu do metod i pól

obiektów na podstawie referencji do nich. Możemy też używać obiektów, których definicji

nie znamy w momencie pisania kodu. Możemy np. pobrać klasę obietu używając `getClass()`

albo tablicę metod używając `getMethods()`.

Dzięki temu program może być modyfikowany w trakcie działa w sposób zależny od

własnego kodu.

13. Czym jest servlet?

Jest to częśc aplikacji po stronie servera, która służy jako most między zapytaniami HTTP

a logiką biznesową aplikacji.

14. Dlaczego metoda `main` jest statyczna?

Żeby JVM mogła ją wywołać bez tworzenia nowego obiektu z klasy ją zawierającej

15. Czym jest wyjątek?

Wyjątek jest to błąd, niespodziewana sytuacja w działaniu programu. Wszystkie wyjątki

dziedziczą na nadklasy `Throwable` i dzielą się na `checked` i `unchecked`.

- `checked`: są sprawdzane w czasie kompilacji i nie pozwalają nam skompilować programu

jeżeli nie będą obsłużone

- `unchecked`: np. RuntimeException albo NullPointery - występują już w trakcie działania

programu

16. Co to jest enkapsulacja?

Jest to proces ograniczania dostępności danych, tak aby dostęp do nich był kontrolowany

przez twórcę programu i możliwy tylko z przewidzianych przez niego miejsc.

17. Różnica między `throw` a `throws`

- `throw` używa się w ciele metody kiedy chcemy wymusić rzucenie danego wyjątku

- `throws` używa się przy definicji metody żeby zaznaczyć że przy korzystaniu z niej

konieczna będzie obsługa tego wyjątku

18. Wyjątki `checked` a `unchecked`

`checked` sa sprawdzane na etapie kompilacji i wymagają obsłużenia przed skompilowaniem programu

`unchecked` występują na etapie działania programu i ich wystąpienie powinno być przewidziane w kodzie

19. Różnice między .jar a .war

.jar służy do przechowywania klas, bibliotek i zasobów potrzebnych do działania aplikacji Java

.war służy konkretnie do przechowywania aplikacji webowych z servletwami i będzie w sobie zawierał

konkretną strukturę danych tzn. WEB-INF itd. War będzie też zawierał widoki, jspeki, javascript itd.

20. Czy możemy mieć obiekt którzy jest immutable zawierający obiekt mutable?

Tak. Przykład poniżej:

//Person jest mutable, bo mamy dostęp do jej pól i możemy je zmieniać

public final class Person(){

public int age;

}

// personRecord jest immutable bo mimo że zawiera mutable obiekt Person

// to nie mamy do niego dostępu

public final class personRecord(){

private final Person person;

}

21. Dlaczego String jest immutable?

a) Bezpieczeństwo

Gdyby String był mutable to możnaby zmienić jego zawartość w trakcie działania metody

na przykład weryfikującej czy nie jest to SQL Injection.

b) Synchronizacja

Dzięki byciu immutable String jest thread-safe i może być dzielony między wielona

threadmi.

c) Cachowanie Hashcode

HashCode dla danego Stringa moze być ustalony tylko raz i używany potem dowolną ilośc razy,

bez ryzyka że zawartość obiektu się zmieni

d) Wydajność

Dzięki istnieniu String Poola (nie mógłby istnieć jeżeli String byłoby mutable)

oszczędzamy miejsce na Heapie i mamy szybszy dostęp do hashy kiedy używamy String.

22. Różnica między stosem a stertą

- Sterta "Heap" to duży obszar pamięci w którym znajdują się stworzone obiekty i referncje

z nich wychodzące. Składa się z trzech częsci: Young Generation, Old Generation, Permanent Generation

Young - tu sa najnowsze obiekty, minor garbage collection czyści tę pamięć i przesuwa obiekty to OG

Old - tu są obiekty które przeżyły kilka mniejszych GC.

Permantn - tu sa trzymane metadane JVM, klasy i metody

- Stos "Stack" to mniejszy obszar pamięci, w którym trzymane są typy proste które są używane

w danym scopie programu. Rozmiar stacka zmnienia się w zależności od tego ile zmiennych

musi pomieścić. LIFO - Last-In-First-Out - kiedy pojawia się nowa zmienna do wskakuje

ona na górę stosu pamięci i jako pierwsza go opuszcza.

23. ArrayList vs LinkedList vs Vector

- ArrayList - rozbudowana tablica, pozwala na dodawanie i odejmowanie obiektów oraz

na dostęp do nich przez podanie ich indeksu. Przy dodaniu obiektu zwięszka rozmiar o 50%.

- LinkedList - implementacja double-linked-list. Każdy obiekt w liście zawiera informacje

o sobie, jak i obiekcie go poprzedzającym i następnym. Lepsza wydajność dodawania i usuwania

niz ArrayList, gorsza wyszukiwania

- Vector - thread-safe ArrayList. Podwaja swój rozmiar przy dodaniu obiektu. Może być przechodzony

iteratorem albo enumeracją

24. HashTable vs HashMap

- HashTable - synchronizowana, nie pozwala na `null` ani jako klucz, ani wartość

- HashMap - pozwala na jeden klucz `null` i nieskończenie dużo wartości `null`

25. Interfejs vs Klasa abstracyjna

Klasa abstracyjna może posiadać metody niedomyślne posiadające logikę biznesową.

Klasa abstrakcyjna może posiadac niestatyczne, niefinalne pola.

ttttterfejsu nie możemy stworzyć obiektu za pomocą new.

Klasy implementujące interfejs musza implementować wszysktie jego metody (oprócz domyślnych).

Klasy moga implementować wiele interfejsów, ale dziedziczyć tylko z jednej klasy abstracyjnej.

26. JIT Compiler (just-in-time-compiler)

JIT Compiler jest częścią JVM odpowiedzialną za compilowanie Java bytecodu do czystego

machine code. Uruchamiany jest przy każdym wywołaniu metody w Javie, kompilując ją do

kodu maszynowego i uruchamiając "Just-In-Time". Wtedy JVM wywołuje ten kod już bez potrzeby

interpretacji bytecode - > machine code, co przyspiesza działanie programu.

27. Local vs Instance variable

- Zmienna lokalna istnieje jedynie w obrębie metody, konstruktora lub bloku kodu i jest usuwana

ze Stacka w momencie opuszczenia przez program danego bloku. Nie jest również dostępna

dla metod spoza tego bloku kodu.

- Instancje variable jest silnie powiązana z klasą w której jest zdefiniowana i jest

jej właściwością. Definiowane są w obrębie klasy ale poza jej metodami.

28. Constructor chaining

Wyołowywanie jednego konstruktora z drugiego. Możemy go używać do wywołania konstruktora

z nadklasy słowem `super()`, lub wywoływać inne konstruktory w obrębie danej klasy używając `this()`

29. Classloader

Część JVM odpowiedzialna na ładowanie klas podczas egzekucji programu. Klasy są ładowane on demand.

Kiedy włączamy JVM używane są trzy classloadery:

- Bootstrap class loader

ładuje podstawowe biblioteki java znajdujące się w <JAVA\_HOME>/jre/lib

- Extensions class loader

ładuje dodatkowe klasy umieszczone w jre/lib/ext albo w katalogach

określonych w właściwości java.ext.dirs

- System class loader

ładuje klasy, które znajdują się w CLASSPATH

30. HashSet vs TreeSet vs LinkedHashSet

- HashSet: dopuszcza jeden null, unsorted, unordered

stały czas podstawowych funkcji

- TreeSet: niedopuszcza nulli, sorted, unordered

czas podstawowych funkcji log(n)

sortowanie w kolejności naturlanej, wzrastającej lub przy użyciu Comparatora

elementy muszą implementować interfejs Comparable, lub trzeba użyć Comparatora

- LinkedHashSet: dopuszcza jeden null, unsorted, ordered

31. Interfejsy i Klasy dostępne w Collections Framework

- Interfejsy:

- Collection

- List

- Set

- Sorted Set

- Navigable Set

- Map

- Sorted Map

- Queue

- Klasy:

- ArrayList

- LinkedList

- Vector

- HashSet

- LinkedHashSet

- TreeSet

- HashMap

- LinkedHashMap

- TreeMap

- HashTable

32. Ordered vs Sorted

- Ordered: kolekcja przetrzymuje dane w kolejności w której zostały dodane

- Sorted: kolekcja sortuje dane na podstawie danego kryterium

33. Priority Queue

- Specjalny typ Queue gdzie wszystkie elementy są ordered używając natural ordering

lub Comparatora użytego przy tworzeniu kolekcji

- Przechowuje obiekty w kolejności rosnącej gdzie czoło kolejki stanowi obiekt o

najniższej wartości

34. Exception propagation

Jeżeli wyjątek nie jest obsłużony zostaje rzucony dalej po stacku aż do momentu

gdzie zostanie obsłużony lub wykrzaczy nam program.

35. Czym jest thread

Thread to ciąg instrukcji wykonywanych przez javę. Kontroluje flow wykonania programu.

Każdy program Java ma przynajmniej jeden thread (main).

Tworzenie nowego Threada odbywa się albo przez implementację interfejsu Runnable

albo przez dziedziczenie z klasy Thread. Dzięki temu musimy zaimplementować

metodę run() w której podajemy przebieg danego wątku, a następnie możemy go wywołać

metodą start().

36. Thread.join()

Metoda służy do synchronizacji wątków. Sprawia że obecny thread będzie czekał na

wykonanie się threadu na którym metoda została wywołana.

37. Thread.yield()

Po wywołaniu daje znać thread scheadulerowi że dany wątek jest gotowy oddać zasoby

jeżeli pojawi się ważniejszy wątek ich potrzebujący

38. Thread.wait()

Zatrzymuje działanie wątku zwalniając wszystkie trzymane przez niego locki.

Obowiązuje aż do wywołania metody notify() lub notifyAll()

39. notify() notifyAll()

Wybudza albo konkretny albo wszytskie wątki w waiting poolu

40. sleep()

Usypia wątek na określoną ilość czasu, ale nie zwalnia trzymanych przez niego locków

41. Life cycle wątków

- New: wątek stworzony, metoda start() nie wywołana

- Runnable: metoda start() wywołana, ale jeszcze nie dotarło do metody run()

- Running: metoda run() w trakcie wykonywania

- Non-runnable (blocked): wątek jest runnable ale jego wykonywanie zostało zablokowane

- Terminated: metoda run() dobiegła końca

42. Transient zmienna

Wartość zmiennej opisanej jako transient nie będzie brana pod uwagę przy serializacji obiektu

43. serialVersionUID

static final long zmienna służaca do identyfikacji zserializowanego obiektu.

44. Czym jest dynamic polymorphism lub dynamic method dispatch

Kiedy wywołanie zoverridowanej metody odbywa się w momencie runtime a nei kompilacji

np:

List<String> list = new ArrayList<>();

list.add("foo");

Po lewej stronie jest interface, po prawej jest jego implementacja

45. Typy dziedziczenia

- Pojedyncze: jedna klasa dziedziczy z jednej klasy

- Multilevel: jedna klasa dziedziczy z klasy, ta klasa z innej klasy itd

- Hierarchical: z jednej klasy dziedziczy kilka klas

- Hybrid: połączenie powyższczych

46. Method hiding

Nie jest możliwe overridowanie metod statycznych lub prywatnych z nadklasy,

kiedy próbujemy to robić to następuje mehtod hiding - ukrywanie że nadklasa

ma w ogóle taką metodę

47. Association, Aggregation, Composition

- Association jest to dowolny związek między dwoma lub wieloma klasami, powiązanie w dowlny sposób.

- Aggregation (has-a) jest to silniejszy typ asocjacji gdzie jeden obiekt przechowuje w sobie

dane o drugim, ale oba mogą istnieć niezaleznie od istnienia drugiego obiektu.

- Composition (is-a-part-of) to najsilniejszy typ asocjacji gdzie usunięcie jednego obiektu

powoduje desktrucję pozostałych

48. Marker interface

Jest to interfejst nieposiadający w ogóle działa, służący jedynie oznaczeniu ze dana

klasa będzie "jakaś" np:

public interface Seriazible(){

}

49. Lazy loading vs eager loading

Lazy loading jest to wzorzec projektowy w którym pełną inicjalizację obiektów odraczamy tak długo,

jak tylko jest to możliwe, co przyczynia się do poprawy wydajności aplikacji.

50. SOLID

- Single responsibility principle

Klasa powinna mieć tylko jedną odpowiedzialność

"A class should have one, and only one, reason to change"

- Open/closed principle

Tworzenie klas tak żeby można było rozszerzać ich funkcjonalność bez nadmiernego

modyfikowania istniejącej już logiki

Open for extension, closed for modyfication.

- Liskov substitution principle

Klasy powinny być tak zbudowane że powinno być możliwe podmienienie obiektów z klas bazowych

na obiekty z klas pochodnych bez wyoływania błędów w programie

- Interface segregation principle

Lepiej tworzyć więcej bardziej szczegółowych interfejsów niż jeden ogólny

- Dependency inversion principle

"high level object should not depend on low level implementations, they should

both depened on abstracions"

51. Jakiej listy użyję żeby dane dodawać np zawsze na początku

Linked list

61. Czy można przekazywać Optional w argumentach metody?

Nie powinno się. Używanie tak Optionali zaciemnia z

apis metody,

sprawia że działanie metody zależy od wartości optionala,

optional w porównaniu do nullowalnych argumentów jest bardziej kosztowny,

Optional może reprezentować aż trzy różne stany: null, non-null ale isPresent = false,

oraz non-null opakowujący właściwą wartość.

62. Skoro nie to jak można obsłużyć zmienną która może być nullem?

Najbardziej eleganckim rozwiązaniem jest stworzenie overloadowanej metody

obsługującej wariant logiki z argumentem null.

63. Jaka lista jeżeli chcę dodawać dane zawsze na początku?

Klasa LinkedList posiada w sobie metody `addFirst()` oraz `addLast()`

64. Co to jest LinkedList?

Podwójnie łączona lista danych - każdy jej node przechowuje informacje

o swojej zawartości, jak i odnośniki do node przed i za nim.

65. Co wprowadziła Java 8?

- Streamy

- Wyrażenia lambda

- Interfejsy funkcyjne

- Try-with-resources

- Optionale

- Typy generyczne

66. Trzy interfejsy funkcyjne i opisać je

- Function: przyjmuje jeden argument i zwraca jeden argument

- Customer: przyjmuje jeden argument i nie zwraca nic

- Predicate: przyjmuje jeden argument i wywołuje na nim test logiczny

- Supplier: nie przyjmuej argumentów, i zwraca jeden

- Operator: przyjmuje i zwraca ten sam typ danych

67. Map vs flatMap w Streamach

- Map służy do mapowania wartości w streamie na inne wartości przy użyciu

interfejsu funkcyjnego Function

- flatMap używamy w momencie kiedy musimy "spłaszczyć" rezultat działania,

tzn: jeżeli mamy np. nested stream, kolekcję lub optional

Optional.of(Optional.of(foo)) lub List<List<String>>

to używając flatMap możemy to spłaszczyć do Optional.of(foo) i List<String>

68. Jak sprawić żeby obiekt był Immutable?

- Ustawić pola jako `private final`

- Gettery zwracające nie bezpośrednią referncję do obiektu, ale jego kopię (klon)

- Ustawić klase jako `final`

- Nie tworzyć setterów do pól

- Inicjalizacja pól z argumentów konstruktora powinna odbywać się

przez deep copy tych arugmentów na pola

69. Jak przekazać do obiektu immutable przez konstruktor np. mapę?

Aby wypełnić immutable pole taką mapą należy jej zawartość skopiować do tego pola

(deep copy)

70. Jak działa HashMapa?

- HashMapa przy inicjalizacji tworzy serię pustych bucketów (domyślnie 16),

z których każdy może mieć jeden lub więcej nodów połączonych jak LinkedList.

- Przy dodawaniu elementu HashMap oblicza hashCode klucza i potem używając

`index = (n-1) & hash` //kalkulacja modulo używając bit mask, szybsze niż `hash % n`

gdzie `n` to ilość bucketów

(klucz `null` wyląduje w bucket 0)

ustala index bucketa, do którego wrzuci dany obiekt.

- Każdy node będzie zawierał:

- key

- hashcode

- value

- next //referencja do następnego node

71. Jak działa HashSet

- HashSet zapewnia unikalność danych, nie zapewniając ani kolejności, ani sortowania.

- Działa to tak, że metoda "add" HashSetu wywołuje metodę "put" istniejącej w nim

HashMapy. Jako klucz dodawany jest Obiekt przeznaczony do dodania do HashSetu,

jako wartość - stały obiekt o nazwie PRESENT.

- Potem metoda `add` sprawdza czy `put` zwrocilo null (udalo sie dodac unikalna wartosc)

czy nie (zwraca PRESENT != null, taka pozycja juz istnieje w mapie) i zwraca

odpowiednio `true` lub `false`

71. a) Czy można dodać do HashMapy dwa klucze o tym samym hashu?

b) Czy możemy zmienić hashcode już istniejącego w mapie obiektu?

c) Co się wtedy stanie?

ad a) Można dodać, trafią wtedy do tego samego bucketa i połączą się w LinkedList.

ab b,c) Możemy, ale nie powinno się tego robić, bo psuje to mapę i obiekt o zmienionym

hashcode nie będzie już dostępny. Najlepiej jeżeli obiekty używane jako klucze

będą immutable.

72. Czy w Javie występuje polimorfizm pól

Nie, o ile metody można overridować to pola w javie są chowane przy dziedziczeniu.

Dzięki temu mamy dostęp i do pól nadklasy przez `super()` i subklasy przez `this()`

73. Bubble sort

Idziemy po kolei po ciągu liczb sprawdzając czy liczba na miejscu `i` jest większa

od liczby na miejscu `i+1`. Jeżeli tak to zamieniamy je miejscami. W ciągu `n-1` iteracji

(gdzie n jest ilością elementów) powinniśmy uzyskać posortowany ciąg.

Przy każdej iteracji największa nieposortowana liczba będzie "płynęła" na koniec ciągu,

ustawiając go w kolejności od najmniejszej do największej.

\* Usprawnienie: dodać flagę `boolean swapped = false` która będzie ustawiana na `true`

jeżeli w danej iteracji nastąpiła zamiana. Jeżeli nie, to można uznać że ciąg

jest już posortowany.

Sprawić żeby algorytm przy każdym przebiegu nie sprawdzał ostatnich, wiadomo że posortowanych

elementów ciągu.

74. Wybrane wzorce projektowe

\* Wzorce kreacyjne

- Singleton

Zapewnia że będzie istniała tylko jedna instancja danego obiektu na ciąg działania

programu. Obenie uważany za antywzorzec.

1. Prywatny konstruktor, żeby uniemożliwić tworzenie większej ilości obiektów

2. `public static final` pole o klasie takiej samej jak singleton

3. `public static` metoda która zwraca instancję klasy

- Fabryka

Fabryka polega na oddelegowaniu zadania tworzenia obiektów z głównej logiki programu

na specjalną klasę. Klasa ta będzie generowała nowe obiekty w zalezności od

przekazanych jej parametrów. Fabryka może być np. Singletonem, lub posiadać statyczne

metody odpowiedzialne za tworzenie obiektów.

- Abstrakcyjna fabryka

"Fabryka fabryk". Wzorzec polega na tym, żeby tworzyć wiele implementacji abstrakcji

odpowiedzialnej za tworzenie obiektów i potem fabryka zwraca nam te implementacje,

czyli konkretne fabryki odpowiedzialne za tworzenie obiektów rządnego typu.

- Builder

Pattern służący do tworzenia obiektów o dużej ilości opcjonalnych pól. Po części

rozwiązuje problemy, jakie mają z nimi fabryki. Builder pozwala tworzyć obiekt

krok po kroku, wypełniając każde pole oraz posiada metodę zwracającą gotowy obiekt.

- Prototyp

Wzorzec używany kiedy tworzenie nowych pełnych obiektów kosztuje zbyt dużo czasu i zasobów,

a istnieje już w programi podobny obiekt. Wtedy kopiujemy ten obiekt i modyfikujemy go

w zależności od naszych potrzeb. Obiekt klonujemy metodą `clone()` wczesniej implementując

interfejs Cloneable. Możemy użyć deep lub shallow copy.

\* Wzorce strukturalne

- Adapter

Służy do łączenia dwóch niezwiązanych ze sobą interfejsów. Taki adapter służyć będzie

do konwersji jednego interfejsu w interfejs oczekiwany przez klienta.

\* Wzorce behawioralne

- Chain of Resposibility

Grupa obiektów która między sobą na rozwiązać pewien problem. Jeżeli jeden z tych obiektów

nie może rozwiązać problemu, zostaje on przekazany dalej w łańcuchu.

- Observer

Obiekt posiada "subskrybentów" (obserwatorów) swojego stanu i może ich informować

o zmianach w nim. Subskrybenci nie wysyłają zapytań do obiektu, tak więc

pozwalają mu spokojnie pracować i czekają na wybudzenie przez ten obiekt.

\* Inne

- MVC

Wzorzec polegający na rozdzieleniu logiki aplikacji webowej na odpowiednie części:

- Model: Dane aplikacji, tu umieszczona jest logika odpowiedzialna np.

za komunikację z bazą danych i dostarczenie tych danych do

kontrolera.

- Controller: Część programu służąca za most między zapytaniami i danymi

pobieranymi z Widoku a modelem. Są to często zestawy kontrolerów

obsługujących zapytania RESTowe.

- View: front-end, część programu z która bezpośrednio oddziaływuje użytkownik

aplikacji webowej (jeżeli nie wysyła czystych zapytań RESTowych)