Wykład 1

1) *Omówić znaczenie klas i obiektów? Jaka jest pomiędzy klasa a obiektem?*
Klasa jest szablonem, z którego tworzy się obiekty. Konstruując obiekt, tworzymy egzemplarz klasy. Wszystko jest obiektem posiadającym zadany typ(inaczej: każdy obiekt jest instancją pewnej klasy). Obiekt może posiadać dane wewnętrzne, metody i można go w jednoznaczny sposób odróżnić od wszelkich innych obiektów. Klasa opisuje zbiór obiektów o tej samej charakterystyce(danych składowych) oraz o tych samych zachowaniach.

2) *Na czym polega enkapsulacja?*
Enkapsulacja inaczej hermetyzacja to jedno z założeń paradygmatu programowania obiektowego. Hermetyzacja polega na ukrywaniu pewnych danych składowych lub metod obiektów danej klasy tak, aby były one dostępne tylko metodom wewnętrznym danej klasy lub funkcjom zaprzyjaźnionym. Gdy dostęp do wszystkich pól danej klasy jest możliwy wyłącznie poprzez metody, (gdy wszystkie pola w klasie znajdują się w sekcji prywatnej, lub chronionej) to taką hermetyzację nazywa się hermetyzacją pełną. Przyczyny stosowania hermetyzacji: uodparnia tworzony model na błędy, lepiej oddaje rzeczywistość, umożliwia rozbicie modelu na mniejsze elementy.
Modyfikatory dostępu:
public - definicje są dostępne dla każdego.
private - dostęp do nich posiada jedynie twórca danej klasy wewnątrz jej funkcji składowych.
protected - działa jak private - różnica polega na tym, że klasy dziedziczące z naszej mają dostęp do elementów protected.
domyślny - do składowych pakietowych mogą się odwoływać inne klasy z tego samego pakietu, natomiast po za pakietem są widziane jako prywatne.

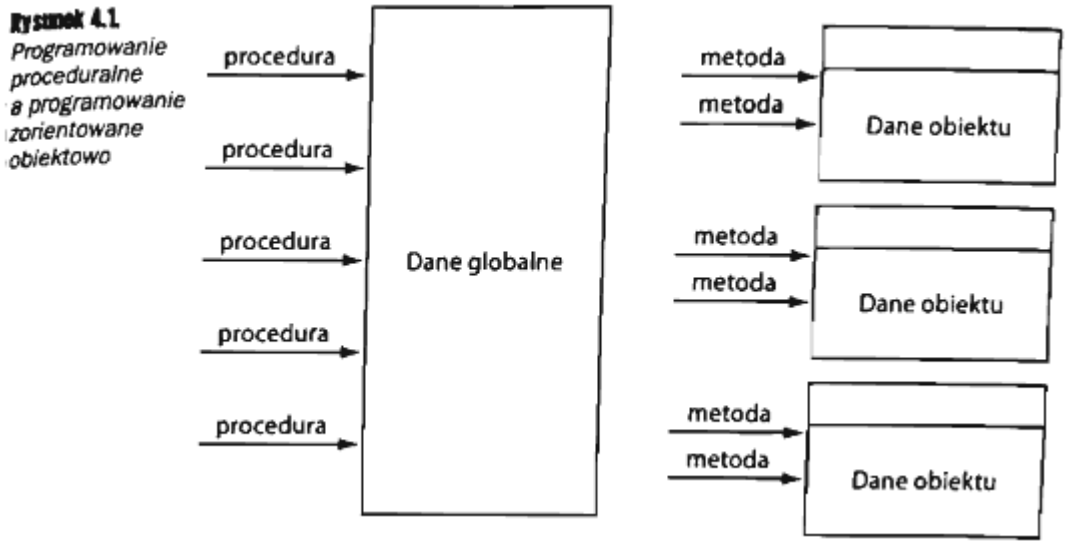
3) *Na czym polega dziedziczenie?*
Dziedziczenie to mechanizm współdzielenia funkcjonalności między klasami, tylko w programowaniu obiektowym. Klasa może dziedziczyć po innej klasie, co oznacza, że oprócz swoich własnych atrybutów oraz zachowań, uzyskuje także te pochodzące z klasy dziedziczonej. Klasa dziedzicząca jest często nazywana *klasą pochodną* lub *potomną* zaś klasa, z której następuje dziedziczenie — *klasą bazową.* Z jednej klasy bazowej można uzyskać dowolną liczbę klas pochodnych. Klasy pochodne posiadają obok swoich własnych metod i pól, również kompletny interfejs klasy bazowej. Klasy pochodne są „kompatybilne w górę" z klasami bazowymi.

4) *Co to jest polimorfizm?*
Polimorfizm (wielopostaciowość) mechanizmy pozwalające programiście używać wartości, zmiennych i podprogramów na kilka różnych sposobów. Inaczej mówiąc jest to możliwość wyabstrahowania wyrażeń od konkretnych typów. Polimorfizm jest kolejnym podstawowym składnikiem języka programowania zorientowanego obiektowo. Dzięki polimorfizmowi uzyskujemy kolejną metodę separacji interfejsu od implementacji, pozwalającą na polepszenie organizacji i czytelności kodu oraz pozwalającej na tworzenie rozszerzalnych programów. Tworzenie programów w wykorzystaniem polimorfizmu pozwala na ich rozwijanie nie tylko podczas powstawania ale również na późniejszym etapie kiedy okaże się że potrzebne jest rozszerzanie go o nowe możliwości.

5) *Na czym polega abstrakcja w programowaniu obiektowym?*
Programowanie obiektowe idzie krok dalej, dostarczając programiście narzędzie do reprezentowania elementów w przestrzeni problemu. Reprezentacja ta jest dostatecznie ogólna, by nie ograniczała się do jakiegoś konkretnego typu problemów. Odwołujemy się do elementów w przestrzeni problemu, jak i do ich odpowiedników w przestrzeni rozwiązania, używając tego samego słowa – obiekt (oczywiście potrzebne są również obiekty, które nie mają odpowiedników w przestrzeni problemu). ***Pomysł polega na umożliwieniu programowi dostosowania się do specyficznego języka danego problemu poprzez dodanie nowych***

typów obiektów, dzięki czemu przy czytaniu kodu opisującego rozwiązanie natrafiamy na słowa wyrażające sam problem. Programowanie obiektowe pozwala więc opisywać problem raczej w kategoriach mu właściwych, nie zaś w kategoriach komputera, na którym zostanie uruchomione rozwiązanie. Każdy obiekt ma swój stan oraz zestaw operacji, które może wykonać, jeśli zostanie oto poproszony.
źródło: Thinking in Java wyd. IV

6) *Najistotniejsze różnice programowania obiektowego i programowania strukturalnego.*
Programowanie strukturalne
1. Programowanie strukturalne jest paradygmatem programowania, zalecającym podział programu na moduły, komunikujące się poprzez dobrze określone interfejsy. Jest rozszerzeniem koncepcji programowania proceduralnego, zalecającego dzielenie kodu na procedury, wykonujące ściśle określone zadania. Procedury nie powinny korzystać z parametrów globalnych, ale przekazywać wszystkie potrzebne dane jako parametry do procedury.
2. Programowanie strukturalne wykorzystuje do rozwiązywania zadań dobrze określone struktury algorytmiczne: sekwencja, selekcja, iteracja i rekursja; unika natomiast stosowania instrukcji skoku.
3. Programowanie strukturalne oddzielnie definiuje dane, oddzielnie funkcje.
Programowanie obiektowe
1. Programowanie obiektowe wprowadza pojęcie obiektu, w którym dane i procedury są ze sobą ściśle powiązane. Program obiektowy korzysta z obiektów, komunikujących się ze sobą w celu wykonania określonych zadań.
2. Cechy typowe dla programowania obiektowego
abstrakcja - zredukowanie właściwości opisywanego obiektu do najbardziej podstawowych; hermetyzacja danych - dostęp do składowych jest ograniczony za pomocą dobrze określonego interfejsu; dziedziczenie - mechanizm umożliwiający wywodzenie nowych klas z klas już istniejących, wraz z przejmowaniem ich metod; polimorfizm - wielopostaciowość, pozwala na wybór metody spośród różnych wersji w zależności od kontekstu.



Rysunek 4.1. Programowanie proceduralne a programowanie zorientowane obiektowo

7) *Typy danych języka Java.*
boolean - (true, false) Boolean; char (16bit) Character; byte (8) Byte; short (16) Short; int (32)

Integer; long (64) Long; float (32) Float; double (64) Double

W Javie podobnie jak w innych językach wyróżniamy wiele typów danych mogących przechowywać zarówno liczby stało i zmiennoprzecinkowe, znaki, ciągi znaków, oraz typ logiczny. Java posiada ścisłą kontrolę typów, czyli mówiąc prościej każdy obiekt musi mieć określony typ.
Liczby stałoprzecinkowe to po prostu liczby całkowite, jednak wyróżniamy ich 4 rodzaje, ze względu na ilość zajmowanego miejsca.

- *byte - 1 bajt – zakres od -128 do 127*
- *short - 2 bajty – zakres od -32 768 do 32 767*
- *int - 4 bajty – zakres od -2 147 483 648 do 2 147 483 647*
- *long - 8 bajtów – zakres od -2^63 do (2^63)-1 (posiadają przyrostek L, lub l)*

Java nie posiada też typu Unsigned (bez znaku), czego konsekwencją jest to, że przekraczając zakres danego typu przejdziemy na zakres ujemny.
liczby zmiennoprzecinkowe:

- *float - 4 bajty – max ok 6-7 liczb po przecinku (posiadają przyrostek F, lub f)*
- *double - 8 bajtów – max ok 15 cyfr po przecinku (posiadają przyrostek D, lub d)*

Kolejnym typem jest char, czyli znak i służy do reprezentacji pojedynczych znaków kodu Unicode. Mogą być przedstawione w znakach pojedynczego cudzysłowu, przy pomocy kodu szesnastkowego, lub po prostu podając numer znaku Unicode w systemie dziesiętnym
Ostatnim typem prostym jest boolean. Reprezentuje on tylko dwie wartości:

- *true - prawda*
- *false - fałsz*

Zazwyczaj jest wykorzystywany jako odpowiednia flaga, lub jako warunek pętli.

To powyżej są to typy proste. W Javie występują również typy obiektowe (każdy typ prosty ma swój obiektowy odpowiednik). Typy obiektowe dostarczają wielu metod dzięki którym możemy np rzutować czy parsować na inne typy. Warto zaznaczyć że tworzenie typu obiektowego trwa dłużej niż typu prostego (typ prosty tworzony jest na stosie a obiektowy na stercie !) . Możliwe jest również automatyczne rzutowanie pomiędzy typami prostymi a obiektowymi np
int x;
Integer y = new Integer (10);
x = y;

To tak z głowy było na podstawie Thinking in Java (poprawiać błędy !)

Wykład 3
***1**. Do czego służą kwalifikatory dostępu?*
<u>Kwalifikator dostępu</u> określa w jaki sposób inne obiekty mogą otrzymać dostęp do danego pola, metody, definicji klasy czy interfejsu. W języku Java wyróżniamy cztery modyfikatory dostępu: publiczny, prywatny, chroniony i domyślny (pakietowy).

***2**. Znaczenie i zastosowanie kwalifikatora private.*
Jest przeciwieństwem modyfikatora publicznego. Pola, metody, klasy i interfejsy oznaczone słowem private są całkowicie niewidoczne poza definicją swojego właściciela. Przykładowa klasa zawierająca zarówno pola, metody, klasy i interfejsy prywatne:
class PrywatnaKlasa {

```
    private int polePrywatne;
    private void metodaPrywatna(){}
}
```

Żaden obiekt nie będzie wstanie odwołać się do zawartości PrywatnaKlasa. Klasa znajdująca się na najwyższym poziomie, czyli PrywatnaKlasa nie może być prywatna.

3. *Znaczenie i zastosowanie kwalifikatora public.*
Oznacza iż do danej metody, pola, definicji klasy i interfejsu ma dostęp każdy obiekt. Dostęp ten nie jest uzależniony o tego w jakim pakiecie znajduje się udostępniana własność i czy to jest ten sam pakiet co pakiet obiektu z którego pochodzi żądanie.

4. *Znaczenie i zastosowanie kwalifikatora protected.*
Modyfikator chroniony oznaczamy słowem kluczowym protected. Elementy oznaczone w ten sposób są widoczne tylko dla innych obiektów znajdujących się poniżej w tej samej hierarchii klas. Innymi słowy dziedziczące po klasie zawierającej elementy chronione. Są też widoczne dla innych elementów znajdujących się w tym samym pakiecie i pod pakietach.

5. *Dostępność klas, pól i metod bez specyfikacji dostępu.*
Modyfikator domyślny można zdefiniować w następujący sposób, jeżeli dostęp do pola, metody, klasy lub interfejsu jest oznaczony jako domyślny to mogą go otrzymać tylko te obiekty, które znajdują się w tym samym pakiecie. Oznacza to, że klasy dziedziczące po klasie zawierającej elementy z dostępem domyślnym nie będą mogły użyć tego elementu chyba, że znajdują się w tym samym pakiecie. Ten sam pakiet oznacza literalnie ten sam katalog w strukturze pakietów. Nie może to być np. podkatalog o innej strukturze pakietów.

6. *Do czego służą pakiety.*
Pakiety to podstawowy mechanizm kontroli dostępu. Służy również do logicznej separacji różnych klas. Java składa się z szeregu klas definiujących obiekty różnego typu. Dla przejrzystości klasy te pogrupowane są w hierarchicznie ułożone pakiety. Każdy pakiet grupuje klasy związane z pewnym szerokim zakresem zastosowań języka np. *java.io* (klasy wejścia-wyjścia). Hierarchię klas oddają nazwy pakietów, które skonstruowane są podobnie jak ścieżki dostępu do plików. Dzięki takiemu systemowi nazwy klas są niepowtarzalne, co pozwala uniknąć niejednoznaczności

7. *Co jest dziedziczone z klasy bazowej?*
Kiedy dziedziczymy mówimy: Ta nowa klasa będzie podobna do tamtej. Składnia wyglądа tak: NazwaKlasy extends NazwaKlasyBazowej. Dzięki temu uzyskujemy wszystkie zmienne i metody składowe klasy bazowej. Stosując dziedziczenie nie jesteśmy ograniczeni jedynie do wykorzystywnia metod klasy bazowej. Możżna równieżż dodawać nowe metody do klasy pochodnej zwyczajnie ją definiując.

8. *Zastosowania dziedziczenia.*
Podstawowym zastosowaniem dziedziczenia jest ponowne wykorzystanie kodu. Jeśli dwie klasy wykonują podobne zadania, możemy utworzyć dla nich wspólną klasę bazową, do której przeniesiemy identyczne metody oraz atrybuty. Ułatwi to testowanie oraz potencjalnie zwiększy niezawodność aplikacji w przypadku zmian.

9. *Zasady inicjalizacji w podklasach i klasach bazowych.*
Najpierw wywoływane są konstruktory klasy bazowej, później klas pochodnych

Wykład 4

1. *Jakie warunki musi spełniać metoda aby mogła być metoda polimorficzna?*
Nie może być zadeklarowana z użyciem modyfikatora final, static lub być prywatną.

2. *Czy klasa abstrakcyjna może zawierać metody ostateczne?*
Tak, lecz nie ma możliwości przesłonięcia tej metody z klasy pochodnej. Metoda taka musi *posiadać ciało.*

3. *Co może zawierać interfejs?*
Metody – domyslnie - public abstract.
Pola – domyslnie - public static final

4. *Czy metody statyczne mogą być ostateczne, abstrakcyjne lub polimorficzne?*
Nie mogą być abstrakcyjne .
Nie mogą być polimorficzne, ponieważ nie mają referencji this.
Mogą być ostateczne, wtedy nie można ich przesłonić.

Wykład 5

1. ***Jakie konstrukcje (klasy, interfejsy, parametry metod, wartość zwracana przez metodę, pole klasy, zmienna lokalna, obiekt) mogą być sparametryzowane typem?***

Możemy:
- *podawać je jako typy pól i zmiennych lokalnych,*
- *podawać je jako typy parametrów i wyników metod,*
- *dokonywać jawnych konwersji do typów oznaczanych przez nie (ale to będzie tylko ważne na etapie kompilacji, po to by uniknąć błędów niezgodności typów, natomiast nie uzyskamy w fazie wykonania faktycznych konwersji np. zawężających, no bo jak?),*
- *wywoływać na rzecz zmiennych oznaczanych typami sparametryzowanymi metody klasy Object (i ew. właściwe dla klas i interfejsów, które stanowią tzw.* ***górne ograniczenia*** *danego parametru typu).*

Nie możemy (w definicjach sparametryzowanych klas i metod):
- *tworzyć obiektów typów sparametryzowanych (new T() jest niedozwolone, no bo na poziomie definicji generics nie wiadomo co to konkretnie jest T),*
- *używać operatora instanceOf (z powodu j.w.),*
- *używać ich w statycznych kontekstach (bo statyczny kontekst jest jeden dla wszystkich różnych instancji typu sparametryzowanego),*
- *używać ich w literałach klasowych,*
- *wywoływać metod z konkretnych klas i interfejsów, które nie są zaznaczone jako górne ograniczenia parametru typu (w najprostszym przypadku tą górną granicą jest Object, wtedy możemy używać tylko metod klasy Object).*

2. ***Jakie sa ograniczenia typów uogólnionych?***

Ograniczenie parametru typu określa zestaw typów, które mogą być używane jako argumenty typu (i podstawiane w szablonie w miejscu parametrów typu), a w konsekwencji zestaw metod, które mogą być wołane na rzecz zmiennych oznaczanych parametrami typu
<U extends Number>
U musi byc klasa dziedziczaca po Number

3. ***Jaka jest róznica pomiedzy typami uogólnionymi a rzutowaniem?***

Typy uogólnione sa konwertowane podczas kompilacji na typy „surowe”.
Samodzielne rzutowanie w dół może być przeprowadzone źle, typy uogólnione sprawiają że nie musimy samodzielnie przeprowadzać rzutowania, są bezpieczniejsze w użyciu.

4. ***Zasady stosowania tablic.***

Deklaracja tablicy składa się z nazwy, typu elementów tablicy, liczby wymiarów tablicy

oraz nazwy zmiennej identyfikującej tablicę. Po ustaleniu rozmiaru tablicy nie może być on zmieniony. Do elementów tablicy odwołujemy się za pomocą indeksów.

Wykład 6

1. Do czego służy i w jakich kolekcjach występuje iterator?

java.util.Iterator – istnieje w każdej kolekcji.
Celem iteratora jest udostępnienie wszystkich elementów kolekcji w sposób niezależny od samej kolekcji i jej implementacji. Dzięki temu w bibliotece Java Collections wszystkie kolekcje można przejrzeć w identyczny sposób. Iterator jest obiektem stanowym: zapamiętuje bieżący element w kolekcji, a wywołanie określonej metody (w Javie jest to next()) powoduje przesunięcie tego wskaźnika na kolejny element. Koniec kolekcji jest określany za pomocą metody hasNext(), która zwraca true tylko wówczas, gdy istnieje jeszcze nieodwiedzony element.

2. Różnice pomiędzy zbiorem a lista.

Zbiór nie może zawierać zduplikowanych elementów, lista może.

3. Jak przegladac wszystkie elementy w mapie?

Należy użyć metody .keySet(), która zwróci nam zbiór wszystkich kluczy i za pomocą pętli foreach
przeglądnąć wszystkie elementy. np.

```
for(String key : aHashMap.keySet()) {
        aHashMap.get(key);
}
```

4. Na czy*m polega identyfikacja typu podczas wykonania?*

Podobnie jak w 5.

5. Kiedy potrzebna jest identyfikacja typu?

Identyfikacja typu potrzebna jest w celu uzyskania dokładnego typu obiektu - czyli w trakcie rzutowania w dół:

```
Ptak p = new Wrobel();
Wrobel w = (Wrobel)p;
```

Żeby móc poprawnie zrzutować w dół, trzeba wiedzieć jakiego typu był dany obiekt.
Uzyskanie informacje w czasie wykonania o typie obiektu to RTTI (run-time type indentification).
Żeby sprawdzić czy dany obiekt jest instancją danej klasy, używamy słowa kluczowego instanceof:

```
if (p instanceof Wrobel) {}
```

Wykład 7

1*. Jakie są wymagania dotyczące wyjątków generowanych przez metody przeciążone lub przesłonięte?*

Metody przeciążone powinny wyrzucać takie same wyjątki. Metody implementujące nie muszą wyrzucać wyjątków specyfikowanych przez metodą abstrakcyjną.

2. *Jakie są zasady specyfikacji generowanych wyjątków w konstruktorach?*
We wszystkich standardowych wyjątkach istnieją dwa konstruktory, pierwszy jest konstruktorem domyślnym, a drugi przyjmuje łańcuch jako parametr tak, że w wyjątku

można umieścić własny komentarz.
W konstruktorach podklas można dodawać nowe wyjątki. Nie należy tworzyć obiektu testowanego w konstruktorze, ponieważ jakikolwiek zgłoszony wyjątek spowoduje przerwanie procesu tworzenia obiektu. Konstruktor klasy pochodnej musi deklarować wszystkie wyjątki konstruktora klasy bazowej. Konstruktor klasy pochodnej nie może przechwytywać wyjątków zgłaszanych przez konstruktor klasy bazowej. Jeżeli wyjątek zostanie zgłoszony wewnątrz konstruktora to garbage collector może nie zadziałać prawidłowo.

***3.** Zasady organizacji wejścia/wyjścia w języku Java.*
Klasy bibioteki I/O w Javie są podzielone według wejścia i wyjście. Przez dziedziczenie wszystkie klasy wyprowadzone z klasy InputStream lub Reader mają podstawowe metody read() i write() służące odpowiednio do odczytania i zapisania jednego lub tablicy bajtów. Metody te jednak rzadko używane są bezpośrednio - istnieją po to, aby mogły skorzystać z nich inne klasy dostarczające bardziej użytecznych interfejsów.
[Coś więcej?]

Wykład 8
1. Jakie warunki musi spełniać klasa aby możliwa była jej serializacja?
[sprawdz to] Musi implementować interfejs Serializable. Pola transient oraz static nie są serializowane.

***2.** Na czym polega serializacja i deserializacja obiektów?*
Istniejący w Javie mechanizm serializacji obiektów pozwala na zmianę dowolnego obiektu, który implementuje interfejs Serializable na sekwencję bajtów w sposób który umożliwia jego wierne odtworzenie w późniejszym czasie. Mechanizm ten działa również w sieci, co oznacza że automatycznie kompensuje różnice pomiędzy systemami operacyjnymi. Co za tym idzie obiekt stworzony np. w systemie operacyjnym Windows po przesłaniu przez sieć do maszyny uniskowej zostanie poprawnie odtworzony. Mechanizm ten zwalania nas z konieczności zadbania o reprezentację danych w różnych systemach, uporządkowanie bajtów czy też inne szczegóły. Serializacja umożliwia zaimplementowanie lekkiej trwałości, co oznacza że czas życia obiektu nie ogranicza się jedynie do czasu życia programu. Dzięki zapisaniu zserializowanego obiektu na dysku pozwala na odtworzenie go przy ponownym uruchomieniu programu, co powoduje uzsykanie efektu trawłości. Dzięki serializacji dane można przechowywać nie tylko w postaci binarnej ale również znakowej (np. XML).
Procesem odwrotnym do serializacji jest deserializacja. Proces ten polega na odczytaniu wcześniej zapisanego strumienia danych i odtworzeniu na tej podstawie obiektu klasy wraz z jego stanem bezpośrednio sprzed serializacji.

***3.** Cechy typu wyliczeniowego.*
Słowo kluczowe enum pozwala na tworzenie nowego typu z ograniczonym zestawem wartości nazwanych i traktowanie tych wartości jako zwykłych komponentów programu.
przykład: enum Kierunek { GÓRA, DÓŁ, LEWO, PRAWO}
Wszystkie stałe wyliczeniowe można przejrzeć jako tablicę za pomocą metody values() (kolejność jak zadeklarowano). Klasa enum implementuje interfejs Comparable.

4. Czy można usunąć/zmienić wartości typu wyliczeniowego?

Nie mozżna zmieniać wartosści typu wyliczeniowego.

Wykład 9
1. Które metody są wystarczające w aplecie?
[Sprawdźcie!] init();
Init() – inicjalizacja – przeglądarka napotykając na tę metodę tworzy wszystkie obiekty, jakie każe stworzyć ta metoda (kolor tła apletu, przyciski)
Start() – startująca – wykonywana po inicjalizacji lub wywoływana, gry użytkownik wróci na stronę z apletem (może być wykonana wielokrotnie)
Stop() – zatrzymująca – wywołana, gdy użytkownik opuści stronę z apletem, lub po wywołaniu metody stop()
Destroy() – koniec życia apletu – zwolnienie pamięci zajmowanej przez program

2. Na czym polega programowanie zdarzeniowe?
Jest to sposób tworzenia programów komputerowych, który określa sposób ich pisania z punktu widzenia procesu przekazywania sterowania pomiędzy poszczególnymi modułami tej samej aplikacji. Program jest cały czas bombardowany zdarzeniami na które musi odpowiedzieć. Następuję przepływ sterowania który jest całkowicie niemożliwy do przewidzenia z góry. Programowanie zdarzeniowe dominuje na etapie programowania GUI. Wszelkiego rodzaju naciśnięcia myszy, żądania odświeżenia okienka, zdarzenia sieciowe są eventami (zdarzeniami).

3. Co realizuje menadżer rozmieszczenia?
Sposób rozmieszczania komponentów w formatce stosowany w Javie różni się od tego powszechnie stosowanego w systemach GUI. Wszystko tutaj zapisane jest w kodzie, nie ma żadnych „zasobów" kontrolujących ułożenie komponentów. Sposób ułożenie komponentów na formatce jest sterowany przez tak zwanego menadżera ułożeń, które decyduje jak komponenty mają być rozmieszczone kierując się kolejnością ich dodawania. Dodatkowo menadżery ułożenia dostosowują wymiany apletu do okna aplikacji, więc jeśli rozmiar okna zostanie zmieniony, to rozmiar, kształt oraz położenie obiektów również może się zmienić.

4. Zastosowanie dziedziczenia w tworzeniu GUI.
[Pytanie 5]

5.
Zastosowanie polimorfizmu w tworzeniu GUI.
Hierarchia klas może przekładać się na hierarchię typów. Możliwe jest wtedy podstawienie pod zmienną (lub atrybut funkcji) typu T obiektu typu S będącego podtypem T i dalsze używanie go jakby był typu T. Jest to możliwe dzięki temu, że podklasa posiada kompletny interfejs swojej nadklasy. W podklasie może być zdefiniowana metoda już istniejąca w nadklasie. Konstrukcja taka umożliwia wykonywanie operacji na obiektach bez informacji, z jakim właściwie obiektem mamy do czynienia. Rozpatrzmy typową aplikację GUI wyświetlającą na ekranie różne komponenty (np. przycisk, pole tekstowe czy listę rozwijaną). Reagują one na te same zdarzenia: kliknięcie myszką, naciśnięcie klawisza, lecz każdy z nich reaguje inaczej, stosownie do tego czym jest. System obsługi zdarzeń najpierw określa,

który z komponentów powinien obsłużyć zdarzenie, a następnie przekazuje mu je. Dzięki podtypowanie opartym na dziedziczeniu możemy utworzyć wspólną klasę Komponent z metodą obsluzKliknieciеMyszka(), którą będą rozszerzać wszystkie rodzaje komponentów. Pobierając aktywny obiekt, możemy wywołać tę metodę bez zastanawiania się czy dany obiekt jest przyciskiem czy polem tekstowym.