**1.A.**

W Javie klasy abstrakcyjne oraz interfejsy używane są w celu tworzenia abstrakcji.

Klasy abstrakcyjne używamy w celu zdefiniowania klas o charakterze abstrakcyjnym, które są wykorzystywane do dziedziczenia przez inne klasy. Abstrakcyjność klasy znaczy, że nie możemy tworzyć jej obiektów.

Interfejsów używamy w celu zdefiniowania abstrakcyjnych typów, jakie mogą posiadać obiekty. Interfejsy zawierają tylko stałe i deklaracje metod bez implementacji.

Klasy abstrakcyjne używa się do zapewnienia wspólnej implementacji metody dla wszystkich podklas lub zapewnienia domyślnej implementacji. Używamy ich gdy klasy są ze sobą mocno powiązane. Na przykład kiedy jest taka sama część kodu i wspólne zmienne dla wszystkich klas dziedziczących z danej klasy, którą możemy zaimplementować w klasie nadrzędnej.

Interfejsów używa się do zapewnienia tego samego zestawu metod dla klas, które nie są ze sobą mocno powiązane.

Klasy abstrakcyjnej używamy gdy:

* Używamy koncepcji dziedziczenia, ponieważ zapewnia wspólną implementację klasy bazowej dla klas pochodnych.
* Chcemy zadeklarować metody niepubliczne. W interfejsie wszystkie metody muszą być publiczne.
* Chcemy w przyszłości dodać nowe metody. Jeśli dodamy nowe metody do interfejsu, wszystkie klasy, które już zaimplementowały ten interfejs, będą musiały zostać zmienione, aby zaimplementować nowe metody.
* Chcemy stworzyć wiele wersji naszego komponentu. Klasa abstrakcyjna zapewnia łatwy sposób na wersjonowanie komponentów. Aktualizując klasę bazową, wszystkie klasy dziedziczące są automatycznie aktualizowane wraz ze zmianą. Interfejsu nie można zmieniać po utworzeniu, jeśli wymagana jest nowa wersja, trzeba stworzyć zupełnie nowy interfejs.

Interfejsy używamy kiedy:

* Chcemy zapewnić wspólną funkcjonalność klasom niepowiązanym.
* Potrzebujemy użyć wielu dziedziczeń, ponieważ możemy zaimplementować wiele interfejsów.
* Projektujemy małe, zwięzłe fragmenty funkcjonalności. Przy dużych jednostkach funkcjonalnych należy użyć klasy abstrakcyjnej.

**1.B.**

|  |  |
| --- | --- |
| **TABLICA** | **LISTA LINIOWA** |
| Jest zbiorem elementów o podobnym typie danych. | Jest zbiorem obiektów określanych jako węzeł (node), które składają się z dwóch części danych i adresu. |
| Elementy są przechowywane w ciągłej lokalizacji pamięci. | Elementy mogą być przechowywane w dowolnym miejscu lub losowo. |
| Działa z pamięcią statyczną, rozmiar pamięci jest stały i nie można go zmienić w czasie wykonywania. | Działa z pamięcią dynamiczną, rozmiar pamięci można zmienić w czasie wykonywania zgodnie z naszymi wymaganiami. |
| Elementy są od siebie niezależne. | Elementy są od siebie zależne. Każdy węzeł zawiera adres następnego, aby uzyskać dostęp do następnego węzła, trzeba uzyskać dostęp do poprzedniego. |
| Wykonanie dowolnej operacji, np. wstawianie, usuwanie itd. Zajmuje więcej czasu. | Wykonywanie dowolnej operacji np. wstawianie, usuwanie itd. Zajmuje mniej czasu. |
| Szybszy dostęp do dowolnego elementu w tablicy, można uzyskać bezpośredni dostęp za pośrednictwem indeksu. | Wolniejszy dostęp do elementu listy liniowej, gdyż zaczyna on przechodzić od pierwszego elementu. |
| Pamięć alokowana w czasie kompilacji. | Pamięć alokowana w czasie wykonywania. |
| Wykorzystanie pamięci jest nieefektywne, np. jeśli rozmiar tablicy wynosi 6, ale składa się tylko z 3 elementów pozostałe miejsce będzie niewykorzystane. | Wykorzystanie miejsca jest wydajne, ponieważ pamięć może zostać przydzielona lub zwolniona w czasie wykonywania zgodnie z naszymi wymaganiami. |