**1. Kiedy istnieje konieczność zarządzania zasobami za pomocą metody Dispose()?   
Czy metodę Dispose() można przeciążać? Jeśli tak to w jakim celu?**

**2. Wyjaśnij czy oba poniższe programy z punktu a) oraz b) są poprawne? Krótko uzasadnij.**

1. Metoda Dispose() w C# jest używana do zarządzania zasobami niezarządzanymi, takimi jak uchwyty do plików, połączenia do baz danych czy uchwyty do systemowych zasobów Windows. Głównym celem jest zwolnienie zasobów,   
   które nie są automatycznie zarządzane przez system wyrzucania śmieci (Garbage Collector) w .NET.

Metodę Dispose() można i powinno się przeciążać w klasach zarządzających zasobami. Implementując IDisposable, zapewniamy kontrolowane zwalnianie zasobów. Zazwyczaj robi się to, aby:

* zamknąć strumienie (np. pliki),
* zwolnić uchwyty do zasobów systemowych,
* zwolnić pamięć niezarządzaną.

W bardziej rozbudowanych scenariuszach stosuje się wzorzec Dispose   
z parametrem bool (do odróżnienia zwalniania zasobów zarządzanych   
i niezarządzanych).

1. Program a) jest poprawny. Prawidłowe zarządzanie zasobami dzięki IDisposable   
   i using. Program b) działa, ale jest mniej bezpieczny i niepoprawny projektowo.

Powinien implementować IDisposable, aby umożliwić użycie using i automatyczne zamykanie pliku.

**7. Czy sprawdzanie dziedziczenia podczas serializacji następuje automatycznie**

**na poziomie frameworka?**

**8. Wymień jakie są dwa sposoby (podejścia) korzystania z mechanizmu serializacji**

**w odniesieniu do konkretnych klas w języku C#?**

7. W przypadku JSON (System.Text.Json) dziedziczenie nie jest automatycznie uwzględniane. Serializator traktuje obiekt jako typ bazowy i nie zachowuje informacji   
o rzeczywistym typie (Student). Żeby obsłużyć poprawnie dziedziczenie w JSON,   
trzeba zastosować bardziej zaawansowane podejścia, np. własne konwertery (JsonConverter) albo dodać metainformacje o typie.

W przypadku XML (XmlSerializer) dziedziczenie również nie jest automatycznie obsługiwane.Aby umożliwić serializatorowi poznanie klas pochodnych, należy dodać atrybut [XmlInclude(typeof(Student))] w klasie bazowej Osoba.

Bez tego serializacja obiektów typu Student zakończy się błędem lub utratą danych.

8. Dwa główne podejścia:

1. Serializacja atrybutowa (dekoratory w kodzie klasy):

- oznaczasz klasy i ich pola specjalnymi atrybutami ([XmlElement], [JsonPropertyName], [XmlInclude], itd.)

- serializator automatycznie odczytuje, jak zamieniać obiekt na XML/JSON i odwrotnie

- stosowane głównie w XmlSerializer, DataContractSerializer i czasami w JSON (Newtonsoft.Json)

2. Serializacja konwencyjna lub konfiguracyjna (przy pomocy ustawień):

- klasa może pozostać czysta (bez atrybutów)

- przy serializacji konfigurujesz zachowanie zewnętrznie, np. przez opcje (JsonSerializerOptions w System.Text.Json)

- przykład: określenie formatu daty, ignorowanie pól, obsługa dziedziczenia bez zmiany kodu klasy