CHAPTER 14

■ ■ ■

Implementing Services

QUESTIONS AND EXERCISES

1. What are services, service interfaces and service providers in Java?

Answer:

A specific functionality provided by an application (or a library) is known as a *service*. The service contains an interface or an abstract class that defines the functionality provided by the service and it is known as *service interface*. Applications and libraries providing implementations of a service are known as service providers.

2. Write the declaration for a module named `M`, which loads service providers of a service interface whose fully qualified name is `p.S`.

Answer:

```
module M {
    uses p.S;
}
```

3. Write the declaration for a module named `N`, which provides the implementation of a service interface `p.S`. The fully qualified name of the service implementation class is `q.C`.

Answer:

```
module N {
    provides p.S with q.C;
}
```

4. How many types of services a module can load using the `ServiceLoader` class?

Answer:

There is no limit on the number of types of services a module may load using the `ServiceLoader` class.

5. How many service implementations of a service type a module can provide?

Answer:

There is no limit on the number of service implementations of a service type a module may provide.

6. When do you use the `java.util.ServiceLoader<S>` class?

Answer:

The `java.util.ServiceLoader<S>` class is used to load service providers. Its `load()` method returns an iterator for all service providers of a specific service interface.

7. When do you use the nested `java.util.ServiceLoader.Provider<S>` interface?

Answer:

Use the nested `java.util.ServiceLoader.Provider<S>` interface when you want to iterate over all providers of a specific service type without instantiating all providers. An instance of the `Provider<S>` interface provides the type of the provider and if needed, its `get()` method may be used to instantiate the provider.

8. You can discover and load service providers of a specific type using the `iterator()` method or the `stream()` methods of the `ServiceLoader` class. Which method has better performance when you have to select a service provider based on the name of the service provider implementation class or interface?

Answer:

The `iterator()` method instantiates a provider before returning. When `stream()` method is used, provider is not instantiated until `get()` method is called which is done after filters are applied. So, the `stream()` method is better when a service provider has to be selected based on the name of the service provider implementation class or interface.

9. What are the provider constructor and provider method? If both are available, which one is used when services are loaded from modular JARs?

Answer:

If a service implementation implicitly or explicitly declares a public constructor with no formal parameters, that constructor is called the *provider constructor*.

If a service implementation contains a public static method named `provider` with no formal parameters, this method is called the *provider method*.

When the `ServiceLoader` class is requested to discover and load a service provider, it checks whether the service implementation contains the provider method. If the provider method is found, the returned value of this method is the service returned by the `ServiceLoader` class. If the provider method is not found, it instantiates the service implementation using the provider constructor.

10. What steps would you take while defining a service in JDK9 packaged in a modular JAR that should also work when placed in the class path?

Answer:

To ensure a service in JDK9 packaged in a modular JAR also works when placed on the class path:

- List fully qualified names of all the service provider implementation classes/interfaces [each class in separate line] in a file with filename being fully qualified service interface name. Place this file in META-INF/services directory.
- Make sure there is a public no-args constructor in every service implementation class.