Quiz yourself: Service types and service providers in Java modules

The ServiceLoader class is key to using the Java Platform Module System.

Which statements are correct about the Java Platform Module System (JPMS)? Choose two.

A. The service type must be a Java interface.

B. The service provider type must be a concrete public Java class.

C. The programmer has an option to inspect the service provider type before obtaining an instance.

D. The service provider type may be a Java interface.

Answer. Option A is incorrect. The service type is commonly an interface, but it may also be a class (either abstract or concrete).

Option B is incorrect, because the service provider type may also be an interface or a class, either abstract or concrete. The reason that abstract types can be used here is that a `public static provider()` method can be provided to serve as a factory. If this method is present, it will be used by the `ServiceLoader`.

There are two possibilities concerning the relationship between the service type and the service provider type.

- If the service provider type has a `public static provider()` method, the service provider type need not have any relationship to the service type, but the return type of the `provider()` method (which will be the actual service implementation, in this case) must be assignment-compatible to the service type.
- If no `provider()` method exists, the service provider type must have a public zero-argument constructor, and that service provider type itself must be assignment-compatible to the service type.

Option C is correct. A stream drawn from the `ServiceLoader` provides a means to find and inspect implementation classes without having instantiated those services. This allows lazy loading such that there is no need to load service implementations that do not appear to be appropriate.

For example, assume a project's code uses `@com.lang.Online` and `@com.lang.Offline` annotations to describe the operating mode of a particular translator factory provider. Therefore, you might have a translator factory class that looks like the following:

```
@Online
public class GoogleTranslatorFactory implements TranslatorFactory {
   ... // code here
}
```

The following code will instantiate only those translator factories that carry the `@Online` annotation:

```
ServiceLoader<TranslatorFactory> loader = ServiceLoader.load(TranslatorFactory.class);
Set<TranslatorFactory> onlineTF = loader
   .stream()
   .filter(p -> p.type().isAnnotationPresent(Online.class))
   .map(Provider::get)
   .collect(Collectors.toSet());
```

Notice that the stream contains `ServiceLoader.Provider` objects; these are a kind of wrapper around the actual service. The `Provider` has two methods: `type()`, which gives access to the java.lang.Class object that describes the service provider, and `get()`, which loads and instantiates that service provider and returns the object. Option D is also correct. As mentioned above, the provider type can be an interface; in that case, the `public static provider()` method must be present and must return an object that's assignable to the service type.

Conclusion. The correct answers are options C and D.