The Java Supplier Functional Interface: Optional Usage

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the Supplier functional interface in Java & recognize how it can be used in conjunction with lambda expressions & method references

Interface Supplier<T>

Type Parameters:

`T - the type of results supplied by this supplier`

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

```
@FunctionalInterface
public interface Supplier<T>
```

Represents a supplier of results.

There is no requirement that a new or distinct result be returned each time the supplier is invoked.

This is a functional interface whose functional method is `get()`.

See docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html

Learning Objectives in this Part of the Lesson

- Understand the Supplier functional interface in Java & recognize how it can be used in conjunction with lambda expressions & method references
- Know how to apply Java Supplier in a concise example

See github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex12

Learning Objectives in this Part of the Lesson

- Understand the Supplier functional interface in Java & recognize how it can be used in conjunction with lambda expressions & method references
- Know how to apply Java Supplier in a concise example
  - This example showcases the Java collection framework's HashMap class & Optional class

Class Optional<T>

java.lang.Object
    java.util.Optional<T>

```
public final class Optional<T>
extends Object
```

A container object which may or may not contain a non-null value. If a value is present, `isPresent()` will return `true` and `get()` will return the value.

Additional methods that depend on the presence or absence of a contained value are provided, such as `orElse()` (return a default value if value not present) and `ifPresent()` (execute a block of code if the value is present).

This is a value-based class; use of identity-sensitive operations (including reference equality (==), identity hash code, or synchronization) on instances of `Optional` may have unpredictable results and should be avoided.

See docs.oracle.com/javase/8/docs/api/java/util/Optional.html

Overview of the Supplier Functional Interface

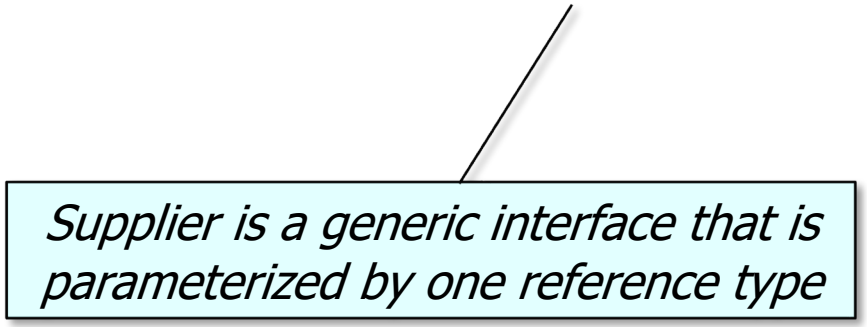
Overview of Supplier Functional Interface

- A *Supplier* returns a value & takes no parameters, e.g.,
  - `public interface Supplier<T> { T get(); }`

See docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html

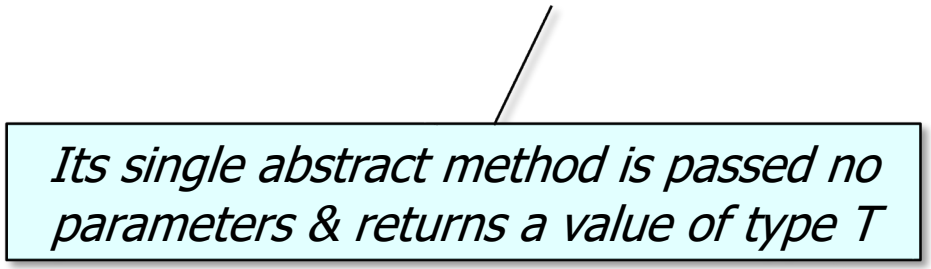
Overview of Supplier Functional Interface

- A *Supplier* returns a value & takes no parameters, e.g.,
  - `public interface Supplier<T> { T get(); }`



Overview of Supplier Functional Interface

- A *Supplier* returns a value & takes no parameters, e.g.,
  - `public interface Supplier<T> { T get(); }`



Applying the Supplier Functional Interface

Applying the Supplier Functional Interface

- This example applies the Supplier functional interface in conjunction with the Java Optional class to print a default value if a key is not found in a Map

```
Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "
                   + being + " = "
                   + disposition.orElseGet(() -> "unknown"));
```

See github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex12

Applying the Supplier Functional Interface

- This example applies the Supplier functional interface in conjunction with the Java Optional class to print a default value if a key is not found in a Map

```
Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "
                   + being + " = "
                   + disposition.orElseGet(() -> "unknown"));
```

Create a map associating each being with its personality traits

Applying the Supplier Functional Interface

- This example applies the Supplier functional interface in conjunction with the Java Optional class to print a default value if a key is not found in a Map

```
Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "
                   + being + " = "
                   + disposition.orElseGet(() -> "unknown"));
```

Get the name of a being from somewhere (e.g., prompt user)

Applying the Supplier Functional Interface

- This example applies the Supplier functional interface in conjunction with the Java Optional class to print a default value if a key is not found in a Map

```
Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "
                   + being + " = "
                   + disposition.orElseGet(() -> "unknown"));
```

Return an optional describing the specified being if non-null, otherwise returns an empty Optional

See docs.oracle.com/javase/8/docs/api/java/util/Optional.html#ofNullable

Applying the Supplier Functional Interface

- This example applies the Supplier functional interface in conjunction with the Java Optional class to print a default value if a key is not found in a Map

```
Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "
                   + being + " = "
                   + disposition.orElseGet(() -> "unknown"));
```

A container object which may or may not contain a non-null value

See docs.oracle.com/javase/8/docs/api/java/util/Optional.html

Applying the Supplier Functional Interface

- This example applies the Supplier functional interface in conjunction with the Java Optional class to print a default value if a key is not found in a Map

```
Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "
                   + being + " = "
                   + disposition.orElseGet(() -> "unknown"));
```

Returns value if being is non-null

See docs.oracle.com/javase/8/docs/api/java/util/Optional.html#orElseGet

Applying the Supplier Functional Interface

- This example applies the Supplier functional interface in conjunction with the Java Optional class to print a default value if a key is not found in a Map

```
Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "
                   + being + " = "
                   + disposition.orElseGet(() -> "unknown"));
```

Calls the supplier lambda value if being is not found

Applying the Supplier Functional Interface

- This example applies the Supplier functional interface in conjunction with the Java Optional class to print a default value if a key is not found in a Map

```
Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "
                   + being + " = "
                   + disposition.orElseGet(() -> "unknown"));
```

orElseGet() uses a "lazy" supplier lambda param

Applying the Supplier Functional Interface

- This example applies the Supplier functional interface in conjunction with the Java Optional class to print a default value if a key is not found in a Map

```
Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "
                   + being + " = "
                   + disposition.orElse("unknown"));
```

Could also use orElse()

See docs.oracle.com/javase/8/docs/api/java/util/Optional.html#orElse

Applying the Supplier Functional Interface

- This example applies the Supplier functional interface in conjunction with the Java Optional class to print a default value if a key is not found in a Map

```
Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

Optional<String> disposition =
  Optional.ofNullable(beingMap.get(being));

System.out.println("disposition of "
                   + being + " = "
                   + disposition.orElse("unknown"));
```

orElse() uses an "eager" value param

Applying the Supplier Functional Interface

- It's also possible to use the getOrDefault() method on Map to accomplish the same behavior without using any Optional features

```
Map<String, String> beingMap = new HashMap<String, String>()
{ { put("Demon", "Naughty"); put("Angel", "Nice"); } };

String being = ...;

String disposition = beingMap
  .getOrDefault(being, "unknown"));

System.out.println("disposition of "
                   + being + " = "
                   + disposition);
```

See docs.oracle.com/javase/8/docs/api/java/util/Map.html#getOrDefault

How Optional Uses the Supplier Functional Interface

How Optional Uses the Supplier Functional Interface

- The Java Optional class uses the Supplier interface in its orElseGet() method
  - `public interface Supplier<T> { T get(); }`

```
class Optional<T> {
  ...
  public T orElseGet(Supplier<? extends T> other) {
    return value != null
      ? value
      : other.get();
  }
```

See docs.oracle.com/javase/8/docs/api/java/util/Optional.html#orElseGet

How Optional Uses the Supplier Functional Interface

- The Java Optional class uses the Supplier interface in its orElseGet() method
  - `public interface Supplier<T> { T get(); }`

```
class Optional<T> {
  ...
  public T orElseGet(Supplier<? extends T> other) {
    return value != null
      ? value
      : other.get();
  }
```

`() -> "unknown"`

The string literal "unknown" is bound to the supplier lambda parameter

How Optional Uses the Supplier Functional Interface

- The Java Optional class uses the Supplier interface in its orElseGet() method
  - `public interface Supplier<T> { T get(); }`

```
class Optional<T> {
  ...
  public T orElseGet(Supplier<? extends T> other) {
    return value != null
      ? value
      : other.get();
  }
```

`() -> "unknown"`

`"unknown"`

The string "unknown" is returned by orElseGet() if the value is null

End of the Java Supplier Functional Interface: Optional Usage