Java 8 Functional Interfaces

Supplier

Douglas C. Schmidt
Learning Objectives in This Lesson

• Recognize foundational functional programming features in Java 8, e.g.,
  • Lambda expressions
  • Method & constructor references
  • Key functional interfaces
    • Predicate
    • Function
    • BiFunction
    • Supplier

```java
@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()`.
```
Overview of Functional Interfaces: Supplier

Douglas C. Schmidt
Overview of Common Functional Interfaces: Supplier

- A Supplier returns a value & takes no parameters, e.g.,
  - public interface Supplier<T> { T get(); }
Overview of Common Functional Interfaces: Supplier

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

*Supplier is a generic interface that is parameterized by one reference type*

See [docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html](http://docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html)
A \textit{Supplier} returns a value & takes no parameters, e.g.,

\begin{verbatim}
public interface Supplier\langle T \rangle {
    T get();
}
\end{verbatim}

Its single abstract method is passed no parameters & returns a value of type \textit{T}.

See \url{docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html}
Overview of Common Functional Interfaces: Supplier

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

```java
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/LiveLessons/tree/master/Java8/ex6](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex6)
Overview of Common Functional Interfaces: Supplier

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

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

  ```java
  String being = ...;
  ```

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

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

Create a hash map that associates beings with their personality traits.
Overview of Common Functional Interfaces: Supplier

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

    ```java
    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"));
    ```
Overview of Common Functional Interfaces: Supplier

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

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 docs.oracle.com/javase/8/docs/api/java/util/Optional.html#ofNullable
A Supplier returns a value & takes no parameters, e.g.,

- public interface Supplier<T> { T get(); }

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 docs.oracle.com/javase/8/docs/api/java/util/Optional.html
Overview of Common Functional Interfaces: Supplier

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

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
Overview of Common Functional Interfaces: Supplier

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

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 supplier lambda value if being is not found
Overview of Common Functional Interfaces: Supplier

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

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

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

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

The string literal “unknown” is bound to the supplier lambda parameter.
Overview of Common Functional Interfaces: Supplier

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

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

The string "unknown" returns by orElseGet() if the value is null.
Overview of Common Functional Interfaces: Supplier

- A Supplier can also be used for a zero-param constructor reference, e.g.,

  ```java
  public interface Supplier<T> { T get(); }
  ```

```java
class CrDemo implements Runnable {
    String mString;

    void zeroParamConstructorRef() {
        Supplier<CrDemo> factory = CrDemo::new;
        CrDemo crDemo = factory.get();
        crDemo.run();
    }

    @Override
    void run() { System.out.println(mString); }
    ...
}
```
Overview of Common Functional Interfaces: Supplier

- A **Supplier** can also be used for a zero-param constructor reference, e.g.,
- public interface Supplier<T> { T get(); }

```java
class CrDemo implements Runnable {
    String mString;

    void zeroParamConstructorRef() {
        Supplier<CrDemo> factory = CrDemo::new;
        CrDemo crDemo = factory.get();
        crDemo.run();
    }

    @Override
    void run() { System.out.println(mString); }
    ...
}
```

Create a supplier that’s initialized with a zero-param constructor reference for `CrDemo`
Overview of Common Functional Interfaces: Supplier

- A Supplier can also be used for a zero-param constructor reference, e.g.,
  
  ```java
  public interface Supplier<T> { T get(); }
  
  class CrDemo implements Runnable {
      String mString;

      void zeroParamConstructorRef() {
          Supplier<CrDemo> factory = CrDemo::new;
          CrDemo crDemo = factory.get();
          crDemo.run();
      }

      @Override
      void run() {
          System.out.println(mString);
      }
  }
  ```

  `get()` creates a CrDemo object using a constructor reference for the CrDemo "default" constructor.
Overview of Common Functional Interfaces: Supplier

• A Supplier can also be used for a zero-param constructor reference, e.g.,
  • public interface Supplier<T> { T get(); }
  class CrDemo implements Runnable {
    String mString;

    void zeroParamConstructorRef() {
      Supplier<CrDemo> factory = CrDemo::new;
      CrDemo crDemo = factory.get();
      crDemo.run();
    }

    @Override
    void run() { System.out.println(mString); } ...
  }

  Call a method in CrDemo to print the result
Overview of Common Functional Interfaces: Supplier

- Constructor references simplify creation of parameterizable factory methods.

```java
public interface Supplier<T> { T get(); }

class CrDemo implements Runnable {
    ...
    static class CrDemoEx extends CrDemo {

        @Override
        public void run() {
            System.out.println(mString.toUpperCase());
        }
    }
    ...
}
```

This class extends CrDemo & overrides its run() method to uppercase the string.

See www.javabrahman.com/java-8/constructor-references-java-8-simplified-tutorial
Overview of Common Functional Interfaces: Supplier

- Constructor references simplify creation of parameterizable factory methods.

```java
public interface Supplier<T> { T get(); }

class CrDemo implements Runnable {
    ...

    static class CrDemoEx
        extends CrDemo {

        @Override
        public void run() {
            System.out.println(mString.toUpperCase());
        }
    }
    ...
```

Print the uppercased value of mString
Overview of Common Functional Interfaces: Supplier

• Constructor references simplify creation of parameterizable factory methods.
  
  public interface Supplier<T> { T get(); }

  class CrDemo implements Runnable {
      ...
      void zeroParamConstructorRefEx() {

      Supplier<CrDemo> crDemoFactory = CrDemo::new;
      Supplier<CrDemoEx> crDemoFactoryEx = CrDemoEx::new;

      runDemo(crDemoFactory);
      runDemo(crDemoFactoryEx);
  }

  ...

Demonstrate how suppliers can be used as factories for multiple zero-param constructor references
Overview of Common Functional Interfaces: Supplier

- Constructor references simplify creation of parameterizable factory methods.

```java
public interface Supplier<T> { T get(); }

class CrDemo implements Runnable {
    ...
    void zeroParamConstructorRefEx() {
        Supplier<CrDemo> crDemoFactory = CrDemo::new;
        Supplier<CrDemoEx> crDemoFactoryEx = CrDemoEx::new;

        runDemo(crDemoFactory);
        runDemo(crDemoFactoryEx);
    }
    ...
```
Overview of Common Functional Interfaces: Supplier

- Constructor references simplify creation of parameterizable factory methods.
  - `public interface Supplier<T> { T get(); }`

```java
class CrDemo implements Runnable {
    ...
    void zeroParamConstructorRefEx() {
        Supplier<CrDemo> crDemoFactory = CrDemo::new;
        Supplier<CrDemoEx> crDemoFactoryEx = CrDemoEx::new;

        runDemo(crDemoFactory);
        runDemo(crDemoFactoryEx);
    }
    ...
}
```

This helper method invokes the given supplier to create a new object & call its `run()` method.
Overview of Common Functional Interfaces: Supplier

- Constructor references simplify creation of parameterizable factory methods.

```java
public interface Supplier<T> { T get(); }

class CrDemo implements Runnable {
    ...
    <T extends Runnable> void runDemo(Supplier<T> factory) {
        factory.get().run();
    }
    ...
}
```

Use the given factory to create a new object & call its run() method
Overview of Common Functional Interfaces: Supplier

- Constructor references simplify creation of parameterizable factory methods.

```java
class CrDemo implements Runnable {
    ...
    <T extends Runnable> void runDemo(Supplier<T> factory) {
        factory.get().run();
    }
    ...
}
```

This call encapsulates details of the concrete constructor that’s used to create an object!
Arbitrary constructors with params can also be supported in Java 8, e.g.,

```java
public interface Supplier<T> { T get(); }

class CrDemo implements Runnable {
    interface TriFactory<A, B, C, R> { R of(A a, B b, C c); }

    void threeParamConstructorRef() {
        TriFactory<String, Integer, Long, CrDemo> factory = CrDemo::new;

        factory.of("The answer is ", 4, 2L).run();
    }

    CrDemo(String s, Integer i, Long l) {
        mString = s + i + l;
    }
}
```

Custom functional interfaces can be defined for arbitrary constructors with params.

This capability is unrelated to the Supplier interface.
Overview of Common Functional Interfaces: Supplier

- Arbitrary constructors with params can also be supported in Java 8, e.g.,

```java
public interface Supplier<T> { T get(); }

class CrDemo implements Runnable { ...

    interface TriFactory<A, B, C, R> { R of(A a, B b, C c); }

    void threeParamConstructorRef() {
        TriFactory<String, Integer, Long, CrDemo> factory = CrDemo::new;

        factory.of("The answer is ", 4, 2L).run();
    }

    CrDemo(String s, Integer i, Long l) {
        mString = s + i + l; }
    }
```

Create a factory that's initialized with a three-param constructor reference
Overview of Common Functional Interfaces: Supplier

• Arbitrary constructors with params can also be supported in Java 8, e.g.,

```java
public interface Supplier<T> { T get(); }

class CrDemo implements Runnable { ...
    interface TriFactory<A, B, C, R> { R of(A a, B b, C c); }

    void threeParamConstructorRef() {
        TriFactory<String, Integer, Long, CrDemo> factory =
            CrDemo::new;

        factory.of("The answer is ", 4, 2L).run();
    }

    CrDemo(String s, Integer i, Long l) {
        mString = s + i + l; }
```