Interfaces - Rules and syntax

The nifty details about interfaces

Abstract methods

Interfaces mainly declares abstract methods. You don’t have to use the abstract keyword (or even the public keyword), the semicolon gives away that detail:

```
// short version
void print();
// same as
abstract void print();
// even the same as:
public abstract void print();
```

Abstract methods - effect on implementing class

Since methods in interfaces typically are abstract, implementing an interface with one or more abstract classes means that an implementing class needs to really implement (write a concrete version of) the abstract methods.

Otherwise, the implementing class needs to be declared abstract!

Interfaces are not classes

- Interfaces cannot be instantiated using the keyword `new`
- Interfaces cannot declare constructors or instance variables

Default methods

- Were introduced in Java 8
- Helps evolving an interface - without breaking implementing classes
- Are concrete methods
- Are declared using the keyword `default`

```
public default void print() {
  System.out.println("I am a default method");
}
```

- Don’t need to be implemented, but can be overridden

Interfaces can declare constants

- The keywords public static final are implied:

```
public interface Logger {
  public static final String FILE_SEP =
      System.getProperty("file.separator");
  public static final String LOG_FILE =
      FILE_SEP + "tmp" + FILE_SEP + "logfile";
  // or with implied public static final:
  int NUMBER = 13;
}
```

Interfaces can be extended

- Can be extended to subinterfaces using the keyword `extends`
- Can actually extend more than one parent interface
  - Not if more than one parent declares the same default method signature
- Constants are “inherited” - can be used in subinterfaces
- Default methods are also inherited
- Abstract methods are inherited
- Example from the API:

```
public interface List extends Collection {
  //...
}
```

Interfaces can declare static methods

- Since Java 8
- Useful for helper methods (used by default methods)
- Allows interface specific static methods to be declared where they belong and where they make sense (in the interface they are specific to)
- Belong to the interface where they are declared
  - As with all static methods, they are not inherited and must be qualified by the interface name if called from other classes/interfaces
- Are implicitly public - you need to use the keyword `static` though!

Further reading

- https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html
- https://docs.oracle.com/javase/tutorial/java/concepts/interface.html