**Интерфейсы** определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может применить множество интерфейсов.

С помощью интерфейса можно полностью абстрагировать интерфейс класса от его реализации. Мы указываем, что именно должен выполнять класс, но не как это делать. Мы также изолируем определения методов от иерархии наследования.

В интерфейсе можно объявить метод с реализацией по умолчанию. Для этого используется ключевое слово default.

Главной **причиной их введения** было стремление предоставить средства для расширения интерфейсов, не нарушая уже существующий код. Ведь при добавлении нового метода в интерфейс придется его реализовывать во всех использующих классах. Поэтому в литературе его иногда называют методом расширения.

Еще одной причиной стала возможность указать в интерфейсе необязательные методы. Они дают свободу классам выбрать метод, который нужно переопределить.

Методы по умолчанию так же позволяют использовать лямбда-выражения для реализации функциональных интерфейсов.

Если в классе реализуются 2 интерфейса с одинаковым методом по умолчанию, но этот метод не переопределяется в данном классе, то возникает ошибка. Можно ссылаться на конкретную реализацию по умолчанию с помощью ключевого слова *super:*

**имя\_интерфейса.super.имя\_метода()**

Если базовый класс имеет такой же метод, как default метод интерфейса, то берется реализация базового класса. Метод базового класса берется также если метод интерфейса не объявлен default.

Метод в интерфейсе можно объявить статическим, с помощью ключевого слова static. Для вызова статического метода используется имя интерфейса: **имя\_интерфейса.имя\_статического\_метода.** Кроме того, метод static может быть вызван в других методах static и default интерфейса.

Его нельзя вызвать через переменную типа интерфейса.Статические методы из интерфейса не наследуются ни реализующими их классами, ни подчиненными интерфейсами.

Статические методы в интерфейсах позволяют группировать связанные служебные методы без необходимости создавать искуственные служебные классы.

Традиционно существовал подход в использовании класса-компаньона. В дополнение к интерфейсу создавался утилитный класс с очень похожим именем, содержащий статические методы, принадлежащие интерфейсу.  
  
Вы можете найти примеры использования данного подхода прямо в JDK: интерфейс java.util.Collection и сопутствующий ему утилитный класс java.util.Collections.