Functional Interfaces

Functional Interface

Example

```
@FunctionalInterface
public interface Comparator<T> {
    int compare(T o1, T o2);

    // default behavior
}
```

Comparator<T>

Comparator<T>

```java
@FunctionalInterface
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```

Example

```
public class Student {
    private int id;
    private String name;
    private int age;

    public Student(int id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public int getId() {
        return this.id;
    }

    public String getName() {
        return this.name;
    }

    public int getAge() {
```

Example

```
import java.util.Comparator;

public class AgeComparator implements Comparator {
    public int compare(Object o1, Object o2) {
        Student s1 = (Student) o1;
        Student s2 = (Student) o2;

        if (s1.getAge() == s2.getAge()) {
            return 0;
        } else if (s1.getAge() > s2.getAge()) {
            return 1;
        } else {
            return -1;
        }
    }
}
```

Example

```
import java.util.Comparator;

public class NameComparator implements Comparator {
    public int compare(Object o1, Object o2) {
        Student s1 = (Student) o1;
        Student s2 = (Student) o2;

        return s1.getName().compareTo(s2.getName());
    }
}
```

Example

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;

public class ComparatorExample {
    public static void main(String[] args) {
        ArrayList students = new ArrayList();
        students.add(new Student(101, "Vijay", 23));
        students.add(new Student(106, "Ajay", 27));
        students.add(new Student(105, "Jai", 21));

        System.out.println("Sorting by Name...");

        Collections.sort(students, new NameComparator());
        Iterator itr = students.iterator();
        while (itr.hasNext()) {
            Student student = (Student) itr.next();
            System.out.println(student.getId() + " " + studen
        }
```

NOT functional interface `Comparable<T>`

Method

```
public interface Comparable<T> {
    public int compareTo(T o);
}
```

Example

```
class Student implements Comparable<Student> {
    private int id;
    private String name;
    private int age;

    public Student(int id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public int compareTo(Student st) {
        if (this.age == st.age) {
            return 0;
        } else if (this.age > st.age) {
            return 1;
        } else {
            return -1;
        }
    }
```

Example

```
import java.util.ArrayList;
import java.util.Collections;

public class CompareExample {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<Student>();
        students.add(new Student(101, "Vijay", 23));
        students.add(new Student(106, "Ajay", 27));
        students.add(new Student(105, "Jai", 21));

        Collections.sort(students);
        for (Student student : students) {
            System.out.println(student.rollno + " " + student.r
        }
    }
}
```

Predicate<T>

Predicate<T>

```java
@FunctionalInterface
public interface Predicate<T> {
    boolean test(T t);
}
```

Example

```
import java.util.function.Predicate;

public class Program {
    public static void main(String[] args) {
        Predicate<Integer> isPositive = x -> x > 0;

        System.out.println(isPositive.test(5));
        System.out.println(isPositive.test(-7));
    }
}
```

BinaryOperator<T>

BinaryOperator<T>

```
@FunctionalInterface
public interface BinaryOperator<T> {
    T apply(T t1, T t2);
}
```

Example

```
import java.util.function.BinaryOperator;

public class Program {
    public static void main(String[] args) {
        BinaryOperator<Integer> multiply = (x, y) -> x * y;

        System.out.println(multiply.apply(3, 5));
        System.out.println(multiply.apply(10, -2));
    }
}
```

UnaryOperator<T>

UnaryOperator<T>

```
@FunctionalInterface
public interface UnaryOperator<T> {
    T apply(T t);
}
```

Example

```
import java.util.function.UnaryOperator;

public class Program {
    public static void main(String[] args) {
        UnaryOperator<Integer> square = x -> x * x;
        System.out.println(square.apply(5));
    }
}
```

Function<T, R>

Function<T, R>

```java
@FunctionalInterface
public interface Function<T, R> {
    R apply(T t);
}
```

Example

```
import java.util.function.Function;

public class Program {
    public static void main(String[] args) {
        Function<Integer, String> convert = x -> String.valueOf
        System.out.println(convert.apply(5)); // 5 долларов
    }
}
```

Consumer<T>

Consumer<T>

```
@FunctionalInterface
public interface Consumer<T> {
    void accept(T t);
}
```

Example

```
import java.util.function.Consumer;

public class Program {
    public static void main(String[] args) {
        Consumer<Integer> printer = x -> System.out.printf("%d
        printer.accept(600);
    }
}
```

Supplier<T>

Supplier<T>

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

Example

```
import java.util.Scanner;
import java.util.function.Supplier;

public class Program {
    public static void main(String[] args) {
        Supplier<User> userFactory = () -> {
            Scanner in = new Scanner(System.in);
            System.out.println("Введите имя: ");
            String name = in.nextLine();
            return new User(name);
        };
        User user1 = userFactory.get();
        User user2 = userFactory.get();
        System.out.println("Имя user1: " + user1.getName());
        System.out.println("Имя user2: " + user2.getName());
    }
}

public class User {
```