람다식 (Lambda)

인터페이스 란? 클래스가 반드시 구현해야 하는 메서드의 약속을 정의하는 것! (구현할 매서드의 약속)
: 인터페이스는 구현한 다음 사용할 수 있다

함수형 인터페이스 란? 구현해야할 매서드가 한 개만 가지는 인터페이스 이다
람다식 이란? 함수형 인터페이스를 구현하는 간단한 방식이다

라이브러리에서 인터페이스 활용 ?
: 라이브러리를 제공하는 입장에서 결정할 수 없는 코드, 즉 사용자 입장에서 구현해야 하는 코드를 인터페이스로 제공한다
(사용자가 반드시 구현해야 할 매서드 정의) : 라이브러리 개발자가 사용자에게 요구하는 구현을 강제함

인터페이스 활용
1) 느슨한 결합을 위해 활용됨 (다형성 활용함)
2) 라이브러리를 만들 때 사용자가 구현할 코드를 약속으로 정할 때 활용됨

함수형인터페이스 람다식으로 구현하기

```
interface Runnable{
    void run();
}

public class Main {
    public static void main(String[] args) {

        //1) 익명으로 구현하기
        Runnable runnable = new Runnable() {
            @Override
            public void run() {
                System.out.println("Runnable 인터페이스 실행!");
            }
        };
        runnable.run();  // run() 메소드 호출
        //2) 람다식으로 구현하기
        Runnable r2  =()->System.out.println("run!!");
        r2.run();
    }
}
```

오전 실습과제)
Runnable 인터페이스를 구현한 클래스 작성하고 제출하기 (3가지 방법)
함수형인터페이스를 하나 만들고 1)이름있는클래스, 2) 익명클래스 3) 람다식 으로 구현하고 제출하기

Comparator 정렬기준 정하기 – Collctions.sort(리스트, Comparator);
Consumer 반복하면서 해야할 코드 – ArrayList의 forEach(Consumer)
스트림의 매개변수로 활용이 많이 됨

```
import java.util.*;

class Product {
    String name;
    int price;

    Product(String name, int price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public String toString() {
        return name + " - " + price;
    }
}

public class ComparatorExample {
    public static void main(String[] args) {
        List<Product> products = new ArrayList<>();
        products.add(new Product("사과", 1000));
        products.add(new Product("바나나", 800));
        products.add(new Product("망고", 1200));

        // 이름 길이 기준으로 정렬
        Collections.sort(products, new Comparator<Product>() {
            public int compare(Product p1, Product p2) {
                return  p1.price  - p2.price;
            }
        });

        System.out.println(products);
    }
}
```

```
import java.util.*;

class Product {
    String name;
    int price;

    Product(String name, int price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public String toString() {
        return name + " - " + price;
    }
}

public class ConsumerExample {
    public static void main(String[] args) {
        ArrayList<Product> products = new ArrayList<>();
        products.add(new Product("사과", 1000));
        products.add(new Product("바나나", 800));
        products.add(new Product("망고", 1200));

        // 각 상품에 대해 10% 할인 적용

        products.forEach(product -> {
            int discountedPrice = (int) (product.price * 0.9);
            System.out.println(product.name  +  "  -  할인  가격:  "  +
discountedPrice);
        });
    }
}
```

java.util.function 패키지

함수형인터페이스	메서드	설명
java.lang.Runnable	void run()	매개변수도 없고 반환값도 없음
Supplier<T>	T get( )	매개변수가 없고, 반환값만 있음
Comsumer<T>	void accept(T t)	매개변수만 있고 반환값이 없음
Function<T,R>	R apply(T t)	하나의 매개변수를 받아서 결과를 반환
Predicate<T>	Boolean test(T t)	조건식을 표현하는데 사용함 매개변수는 하나 반환 타입은 Boolean

```
public class Main {
    public static void main(String[] args) {
        // Runnable 인터페이스 익명 구현
        Runnable runnable = new Runnable() {
            @Override
            public void run() {
                System.out.println("Runnable 인터페이스 실행!");
            }
        };
        runnable.run();  // run() 메소드 호출
        Runnable r2  =()->System.out.println("run");
        r2.run();
    }
}
```

```
import java.util.function.Supplier;
public class Main {
    public static void main(String[] args) {
        // Supplier 인터페이스 익명 구현
        Supplier<String> supplier = new Supplier<>() {
            @Override
            public String get() {
                return "Supplier 인터페이스 실행!";
            }
        };
        String result = supplier.get();
        System.out.println(result);

        Supplier<String>  s2 =() -> return "문자열 제공" ;
        System.out.prinlns(s2.get());
    }
}
```

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

public class Main {
    public static void main(String[] args) {
        // Consumer 인터페이스 익명 구현
        Consumer<String> consumer = new Consumer<>() {
            @Override
            public void accept(String value) {
                System.out.println("Consumer가 받은 값: " + value);
            }
        };
        consumer.accept("Hello, World!");

        Consumer<String> c2=( str ) -> System.out.println(str);
        c2.accept( "hello consumer");
    }
}
```

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

public class Main {
    public static void main(String[] args) {
        // Predicate 인터페이스 익명 구현
        Predicate<Integer> predicate = new Predicate<>() {
            @Override
            public boolean test(Integer number) {
                return number > 10;
            }
        };

        // Predicate 실행
        int num = 15;
        if (predicate.test(num)) {
            System.out.println(num + " 은(는) 10보다 큽니다!");
        }

        //람다식으로 구현하고 실행해 보자 !
    }
}
```

오후 실습 과제) 1번만, 1,2번 둘 중 하난만 하면 된다 !!!
1) 함수형인터페이스 5개를 스토리를 가지고 구현해 보자
2) 요리주제말고 다른 주제를 가지고 5개 인터페이스 구현해 보자

Runnable : void run ()
Consumer〈T〉 : void accept(T t)
Supplier〈T〉 : T get( )
Predicate〈T〉 : Boolean test (T t)
Function〈T,R〉 : R apply (T t)

##############################
오후 실습과제
##############################
Runnable
Consumer〈T〉
Supplier〈T〉
Predicate〈T〉
Function〈T,R〉

1. Runnable : 버킷리스트 출력하기 – 입력없고 반환없다
2. Consumer : 3만원으로 장보기 (3만원을 매개변수로 받아서 소비한다) – 입력있고 반환없다
3. Supplier : 요리만들기 (레시피 정보 출력하고 반환값은 요리명을 반환해 주세요 !)
–입력없고 반환있다
4. Predicate : 입력으로 받은 요리가 내가만든요리인 경우 true, false반환하기
–입력있고 반환있다 (반환은 무조건 boolean)
5. Function : 입력하나 반환있는 함수 만들기 (2개 이상만 작성하기)
–입력있고 반환있다 (반환타입을 정할 수 있다)
– 입력으로 들어오는 수의 제곱 반환하기
– 입력으로 들어오는 수의 범위안의 난수 반환하기
– 입력으로 들어오는 금액에 대한 화폐매수 구하기

##

라이브러리 만들기

```
public class LibA{


   //별5개를 출력하는 기능
    public void forStar()  {
         for( int i=1; i<=5; i++){
              System.out.println("★");
         }
   }


  // 반복할 문자를 줘 , 매개변수로 값을 전달받을 수 있다.
   public void forCharacter(  char  something ) {
         for( int i=1; i<=5; i++){
              System.out.println( something );
         }
    }



  // 매개변수로 코드를 받아야 한다면
  //  << 코드를 줘 ,  내가 반복해 줄께 >>
  // 매개변수로 무엇을 받아야 하나요?


  // 함수(매서드)가 한개인 인터페이스를 활용한다.
  public void forSomethingDo(     ?   ){
       for( int i=1; i<=5; i++){
                 ?
       }
   }

}
```

```
  interface 코드 {
         public void doing();
  }


public class LibA{

    public void forStar()  {
         for( int i=1; i<=5; i++){
              System.out.println("★");
         }
    }


  // 반복할 문자를 줘
    public void forCharacter(  char  something ) {
         for( int i=1; i<=5; i++){
              System.out.println( something );
         }
    }


  // 반복할 코드를 줘
  // 내가 반복해 줄께
  // 매개변수로 무엇을 받아야 하나요?
  // 라이브러리를 만드는 사람은 무엇을 반복할지는 모른다. 그래서 인터페이스를 사용한다.
  // 해야할 코드를 @FunctionalInterface로  받는다 ( 인터페이스로 받는다 )


  public void forSomethingDo(  코드  somethingDo  ){
       for( int i=1; i<=5; i++){
              somethingDo.doing();
       }
    }
}
```

```
public class TestMain {
        public static void main(String[] args) {
                //2) 인터페이스를 익명클래스로 구현한기
                lib.코드반복실행하기(  new 코드() {
                        @Override
                        public void 실행코드() {
                                System.out.println( "3단  !!!!");

                        }});
                //3.람다식으로 구현하기  (함수형인터페이스만 가능함)
                lib.코드반복실행하기( ()-> {
                        System.out.println( "3단  !!!!");
                });
        }
}
```