Lambda & Functional Programming

functional programming =

functional Thinking

+

functional Coding

What is Functional Programming?

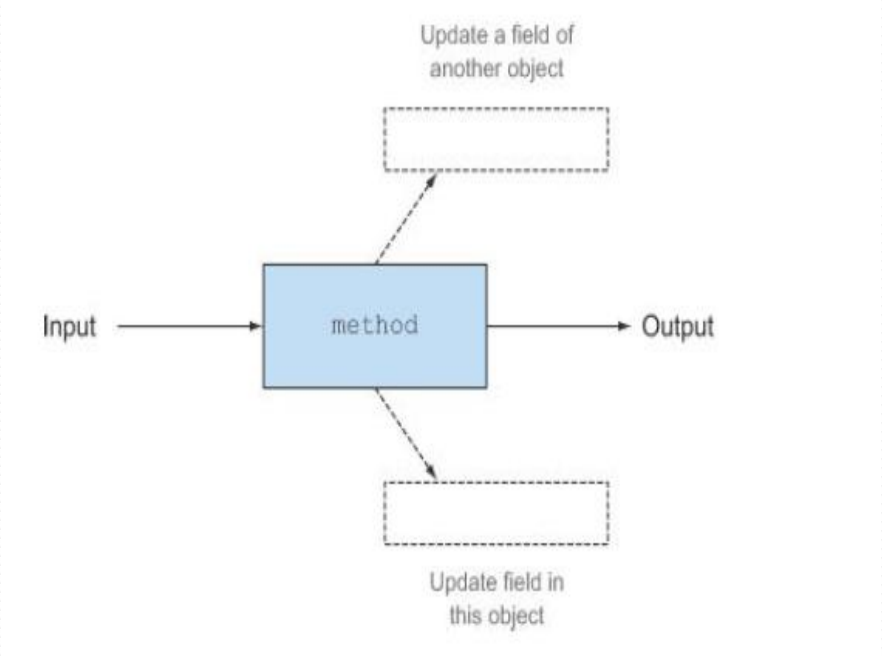
- Higher order functions များ(no-side-effects methods များ)ဖြင့် ရေးသားထားသည့် programming ပင်ဖြစ်သည်။
- method သည် ၎င်းရှိသည့် class အတွင်းမှ state များရော တခြား objects များ၏ state များကိုပါ ပြုပြင်ပြောင်းလဲခြင်း မရှိဘဲ return key-word ကိုအသုံးပြုပြီး ရလဒ်တစ်ခုလုံးကိုသာ return ပြန်သည်ကို pure (သို့မဟုတ်) side-effect free ဟုခေါ်ဆိုသည်။
- Side effect ဆိုသည်မှာ အလုပ်လုပ်ဆောင်မှုများကို function အတွင်းတွင် အလုံးစုံပိတ်ဝှံ့ မထားနိုင်ခြင်းဖြစ်သည်။

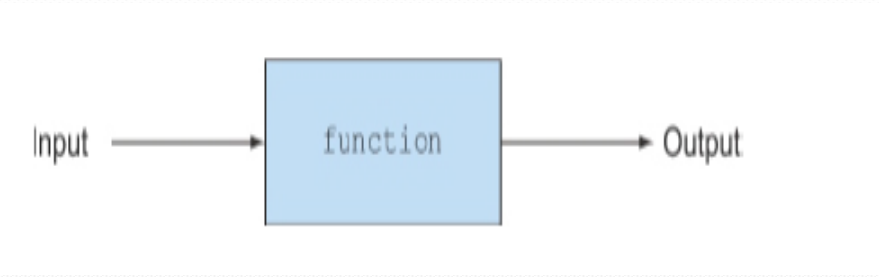
  ဥပမာ။ ။ setter methods ၊

  throwing an exception

  IO operation များကို အလုပ်လုပ်ဆောင်ခြင်း

Side effect & No Side effect methods





အော့နလုံးနာစရာ ဘေးထွက်ဆိုးကျိုးများကို ရှောင်ကြည်ကြ

Benefits of No Side-Effects

- No side effects ကို immutable objects များဟု မှတ်ယူနိုင်သည်။
- ၄င်းတို့ကို object တည်ဆောက်ပြီးသည်နှင့် မည်သည့်အရာမှ မပြုပြင်မပြောင်းလဲနိုင်ပါ။ ကျွန်တော်တို့ မမျှော်မှန်းထားသည့် အခြေအနေများကို မည်သို့မှ မရောက်နိုင်ပါ။
- ၄င်းတို့ကို ပြုပြင်ပြောင်းလဲခြင်းမပြုနိုင်သောကြောင့် thread safe ဖြစ်ကြသည်။
- Multicore parallelism ကို locking ပြုလုပ်ရန်မလိုဘဲအသုံးပြုနိုင်သည်။အကြောင်းမှာ methods များသည် အချင်းချင်း ပင်ရောက်နှောက်ယှက်ခြင်းမရှိကြသောကြောင့်ဖြစ်သည်။
- Program ၏ မည်သည့်အပိုင်းများသည် အမှီအခိုကင်းစွာရပ်တည်နေသည်ကို လျှင်လျှင်မြန်မြန် နားလည်နိုင်လာသည်။

FP ၏ အဓိကအကြောင်းအရာများ

- Declarative programming
- Referential transparency
- Immutability
- Currying
- Persistent data structure
- Pattern matching
- Combinators
- Lazy list တို့ဖြစ်ကြသည်။

Declarative Programming

- FP သည် declarative style ရေးသားပုံအပေါ်တွင် အခြေခံထားသည်။
- How အပေါ်တွင် အခြေမခံဘဲ What ကို အခြေခံထားသည်။
- အလုပ်လုပ်ဆောင်မည့် အရာများကိုသာ သတ်မှတ်ပေးပြီး ၄င်းကို မည်သို့မည်ပုံ ပြီမြောက်အောင်ဆောင်ရွက်ရန်ဆိုသည်ကို system အားရွေးချယ်ဆုံးဖြတ်စေသည်။
- Code ရေးသားခြင်း ၊ ဖတ်ရှုခြင်းများကို သဘာဝကျကျလုပ်ဆောင်သည်။
- Programs များကို ဒီဇိုင်းဆွဲခြင်း ၊ ရေးသားခြင်း ၊ ပြုပြင်ထိန်းသိမ်းရန် ပိုမိုလွယ်ကူ လာသည်။ သို့သော် programmer သည် စက်အပေါ်ထိန်းကွပ်မှုများ နည်းပါးလာ သည်။
- Java တွင် Business Logic များကို အလုပ်လုပ်ဆောင်ရန်ရေးသားရာတွင် သင့်တော် သည်။

Sandwich algorithm

- Imperative
  1. Take a bread
  2. Spread bread with butter
  3. Put cheese on the bread
  4. return result

- Functional Declarative Style

```
return put( cheese,
            spread ( butter, bread) )
```

Currying

- ၂၀ ရာစု သင်္ချာပညာရှင် Haskell Curry က ရေးသားခဲ့သည်။
- Currying means taking a function that takes multiple arguments and truning it into a chain of functions each taking one argument and returning the next function until the last returns the result.
- Unit conversion များတွင် conversion factor နှင့် မြောက်ပြီး baseline နှင့် ညှိပေးရသည်။

```
static double converter(double x, double f, double b){
        return x * f + b;
```

} ကို ပိုမိုကောင်းမွန်သည့် one argument conversion functions သို့ ပြောင်းနိုင်သည်။

```
static DoubleUnaryOpeator curriedConverter(double f,double b){
        return (double x) -> x * f + b;
}
```

Referential transparency

- တူညီသည့် argument value နှင့် ခေါ်သည့်အခါတိုင်း တူညီသည့် ရလဒ်သာ အမြဲ return ပြန်သည်။

  ဥပမာ။ ။ "raoul".replace('r', 'R');

- RT ဥပဒေသကို ချိုးဖောက်မှုများ

  1.Modifying a variables

  2.Modifying a data structure in place

  3.Setting a field on an object

  4.Throwing an exception or halting the error

  5.Printing to the console or reading user input

  6.Reading from or writing to a file

Lambda

- Java သည် pure functional programming မဟုတ်ပါ။အကြောင်းမှာ အချို့ model များသည် referential transparency ဥပဒေကို ချိုးဖောက်နေသောကြောင့် ဖြစ်သည်။ IO မှ methods များကို အသုံးပြုပြီး file ကို နှစ်ကြိမ်ဖတ်ရာတွင် မတူကွဲပြားသည့် ရလဒ်သာရမည်။ Exceptions များကို throws ပြုလုပ်ခြင်းများက RT ဥပဒေသကို မလိုက်နာရာရောက်သည်။
- Java သည် version 8 တွင် Lambda ကို အသုံးပြုပြီး functional style programming ကိုရေးသားနိုင်လာသည်။

Functional Style Programming

STORE functions in variables

PASS functions in parameters

RETURN functions from other functions

What is Lambda?

- Lambda ဆိုသည်မှာ anonymous method ဖြစ်သည်။
- ၄င်း၏ type မှာ functional interface ဖြစ်သည်။
  - functional interface ဆိုသည်မှာ single abstract method (SAM) ပါဝင်သည့် interface ဖြစ်သည်။ @FunctionalInterface annotation ကို အသုံးပြုပြီး functinal interface ဟုတ်မဟုတ် စစ်ဆေးနိုင်သည်။
- Lambda ကို variable ထဲသို့ထည့်သွင်းနိုင်သည်။
- method parameter အဖြစ် အသုံးပြုနိုင်သည်။
- method တွင် lambda ကို return ပြန်နိုင်သည်။

Common functional interfaces in Java 8

Functional interface	Function descriptor	Primitive specializations
Predicate<T>	T -> boolean	IntPredicate, LongPredicate, DoublePredicate
Consumer<T>	T -> void	IntConsumer, LongConsumer, DoubleConsumer
Function<T, R>	T -> R	IntFunction<R>, IntToDoubleFunction, IntToLongFunction, LongFunction<R>, LongToDoubleFunction, LongToIntFunction, DoubleFunction<R>, ToIntFunction<T>, ToDoubleFunction<T>, ToLongFunction<T>
Supplier<T>	() -> T	BooleanSupplier, IntSupplier, LongSupplier, DoubleSupplier
UnaryOperator<T>	T -> T	IntUnaryOperator, LongUnaryOperator, DoubleUnaryOperator
BinaryOperator<T>	(T, T) -> T	IntBinaryOperator, LongBinaryOperator, DoubleBinaryOperator
BiPredicate<L, R>	(L, R) -> boolean	
BiConsumer<T, U>	(T, U) -> void	ObjIntConsumer<T>, ObjLongConsumer<T>, ObjDoubleConsumer<T>
BiFunction<T, U, R>	(T, U) -> R	ToIntBiFunction<T, U>, ToLongBiFunction<T, U>, ToDoubleBiFunction<T, U>

A rich set of functional interfaces:Supplier

- Contains method get that takes no arguments and produces a value of type T.Often used to create a collection object in which a stream operation's result are placed.

  - Supplier

```
@FunctionalInterface
public interface Supplier<T> {

   T get();
}
```

A rich set of functional interfaces:Consumer

- Contains method accept that takes a T argument and return void.Performs a task with it's T argument, such as outputting the object, invoking a method of the object.

  - Consumer

```
@FunctionalInterface
public interface Consumer<T> {

   void accept(T t);
}
```

A rich set of functional interfaces:BiConsumer

```
@FunctionalInterface
public interface BiConsumer<T, U> {

    void accept(T t, U u);
}
```

Functional interfaces:Predicate/BiPerdicate

- Contains method test that takes a T argument and returns a boolean. Tests whether the T argument satisfies a condition.

  - Predicate / BiPredicate

```
@FunctionalInterface
public interface Predicate<T> {

   boolean test(T t);
}
```

```
@FunctionalInterface
public interface Predicate<T, U> {

   boolean test(T t, U u);
}
```

Functional Interfaces:Function/BiFunction

- Contains method apply that takes a T argument and returns a vlaue of type R.Calls a method on the T argument and returns that method's result.

- Function / BiFunction

```
@FunctionalInterface
public interface Function<T, R> {

   R apply (T t);
}
```

```
@FunctionalInterface
public interface BiFunction<T, U, R> {

   R apply (T t, U u);
}
```

Lambda:Behaviour parameterization

- Behaviour parameterization သည် မကြာခနပြောင်းလဲရမည့် အခြေအနေများကို ထိန်းကွပ်ပေးနိုင်သည့် software development pattern ဖြစ်သည်။
- classical design patterns များထဲမှ Strategy pattern သည် Change Conditions များကို handle လုပ်နိုင်သည်။
- Lambda : behaviour parameterization သည် Strategy pattern ကဲ့သို့ပင် အလုပ်လုပ်ဆောင်ပြီး ၊ code ရေးသားရပိုမိုလွယ်ကူခြင်း ၊ ပိုမိုဖတ်ရှုနားလည်နိုင်ပြီး ၊ ကျစ်လစ်သည့် code များကို ဖြစ်စေသည်။
- Philosophy:Forbidden Modification, Open for Extensions…
- နမူနာများ

Lambda:Better Version of Anonymous Inner Class

- Anonymous inner class နေရာတွင် အစားထိုးအသုံးပြုနိုင်သည့် ပိုမိုကောင်းမွန်သည့် ပုံစံဖြစ်သည်။
- Anonymous inner class ထက်ပိုမို ရေးသားရလွယ်ကူပြီး ၊ code ဖတ်ရှုရာတွင်ပိုမို လွယ်ကူလာသည်။
- နမူနာများ

Lambda make Behaviour Abstraction

- OOP ၏ Encapsulation ဖြင့် state များကို abstraction ဖြစ်စေခဲ့သည်။
- Java 8 တွင် lambda ကို အသုံးပြုပြီး method များကိုပါ abstraction ဖြစ်လာသည်။
- နမူနာများ

Beyond Java 8

- Java 8 features များသည် programming style ကို များစွာပြောင်းလဲလာစေသည်။
- Lambda သည် Java ၏ နေရာတိုင်းတွင် ပုံနှံ့နေရာယူစပြုလာသည်။

ဥပမာ။ ။

1.Collections and Maps with Lambdas

2.I/O with Lambdas

3.Data Access with Lambdas

4.Lambda Concurrency

5.Functional Style with Lambdas

6.Streams Api with Lambda

- Parallel အလုပ်လုပ်ဆောင်ရန်လိုအပ်သည့် code လွယ်ကူရိုးရှင်းစွာပင် multicore-processors များ၏ အစွမ်းကို အပြည့်အဝအသုံးပြုလာနိုင်သည်။
- Data များကို processing လုပ်ဆောင်ရန် declarative style ကိုထောက်ပံ့လာမှုများသည် OOP မှ မရရှိသောစွမ်းဆောင်ရည်များဖြစ်သည်။

Conclusion:

- လွန်ခဲ့သည့် 10 နှစ်အတွင် အကြီးအကျယ်ပြောင်းလဲမှုသည် အမှန်တွင် အစပြုခြင်းသာဖြစ်သည်။
- Java 8 သည် ခနနားရန် ကောင်းမွန်သည့်နေရာသာဖြစ်သည်။ရပ်နားရန်နေရာမဟုတ်ပါ။
- Functional Programming ကို Java 8 အစပြုပေးပြီး နောင်လာမည့်အနာဂတ်တွင် ပိုမိုကောင်းမွန်ပြည့်စုံလာမည်မှာ အသေအချာပင်ဖြစ်သည်။

Thank You!