JAVA 2

Enum, Generics

Teme

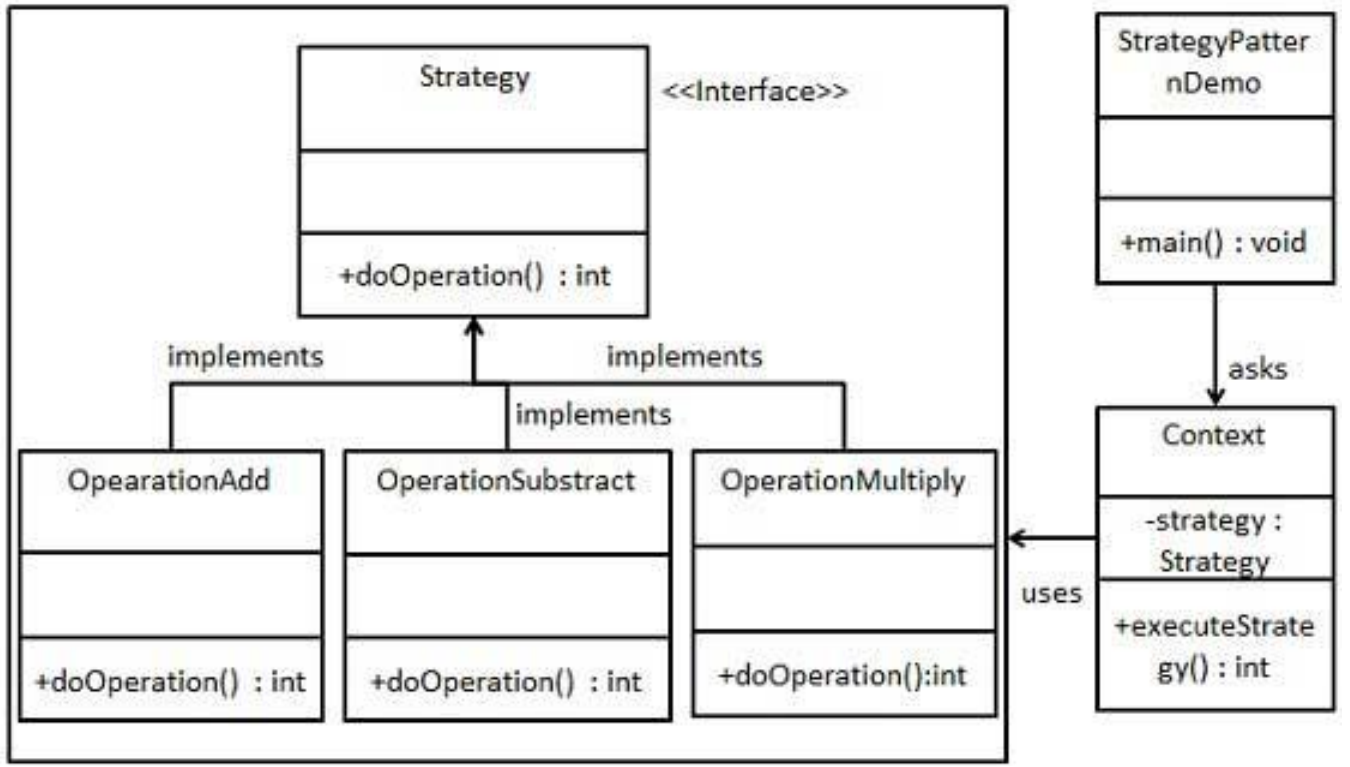
- Enum
- *Strategy pattern*
- Generics
- *Array vs Generics*
- *Generics* ograničenja
- PECS

Enum

- referentni tip
- sadrži pobrojani niz konstanti
- pogodan za *switch-case*
- konstante su implicitno *final* i *static*
- kreiranje instanci jedino tijekom deklaracije
- može sadržavati konstruktore, varijable…
- values() – dohvaća sve vrijednosti enumeracije
- implicitno najbolji *Singleton* (Java1), ovdje *Strategy Pattern*

Strategy pattern

- ponašanje (algoritam) klase može se promijeniti tijekom izvršavanja
- bihevioralan



Izvor: https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm

Generics

- poput *template* u C++*,* ali ne rade sa primitivima
- J2SE 5 – razlog za *Wrapper* klase, *Autoboxing*, *Unboxing*
- omogućavaju apstrakcije nad tipovima – izbjegavanje redundancije
- parametriziranje metoda (parametri i povratne vrijednosti), klasa, sučelja
- generiraju *compile-time* umjesto *run-time* greške
- moguće koristiti više tipova (T, U, R)
- moguće ograničavanje tipova (*extends*, *super*)
- kontejneri – *Collection Framework*

Array vs Generics

Integer extends Number

- *Array*
  - polimorfizam - *Number[]* može sadržavati *Integer*
  - *covariant – Integer[]* je podklasa *Number[] – possible runtime exception*

```
Integer[] ints = {1, 2, 3};
Numbers[] nums = ints;
nums[0] = Math.PI;
```

  - *reifiable – runtime* točno zna da je *nums Integer[]* i baca *exception*
  - prevarili smo *compiler,* ali ne i *runtime*

Array vs Generics

Integer extends Number

- *Generics*
  - polimorfizam – *List<Number>* može sadržavati *Integer*
  - *invariant* – *List<Integer>* nije podklasa *List<Number>*
  - *non-reifiable* – *runtime* ne zna da je *List* tipiziran na *Integer*, radi *backward-compatibility*
  - *type erasure* – *type* parametri se nakon kompajliranja *izbacuju*
  - *runtime* – siguran, jer se svi problemi uočavaju u *compile time*
  - kako povratiti fleksibilnost koju je imao *Array?*
- *backward compatibility* sa *raw types* - napustiti *raw types* jer gubimo sve benefite ove zaštite!

Generics ograničenja

- *Holder<T>*
  - *raw Holder* može sadržavati bilo koji *Object – ipak nije kao Holder<Object>!*

Holder<Integer> ints = new Holder<Integer>(5);

1.)

unsafeSet(ints, "Milica");

*private static void unsafeSet(**Holder holder**, Object value) {*

*   **holder.setValue(value); // runtime exception***

}

2.)

***unsafeSet(ints, "Milica"); // compile time error** – invariace (~~Holder<Number> extends Holder<Object>~~)*

*private static void unsafeSet(**Holder<Object> holder**, Object value) {*

*   holder.setValue(value);*

}

Generics ograničenja

- *Holder<T>*
  - *Holder<Integer>* - sigurni smo

unsafeSet(ints, "Milica");

*private static void unsafeSet(**Holder<Integer> holder**, Object value) {*

*	**holder.setValue(value); // compile time error*** *- Holder<Integer>* može primiti samo *Integer*

}

  - <?> - *unbounded wildcard – ANY* (bilo koji tip) – *read only*

unsafeSet(ints, "Milica");

private static void unsafeSet(Holder<?> holder, Object value) {

*	**holder.setValue(value); // compile time error - Holder<?> može samo čitati, ali kao Object***

*	Object o = holder.getValue();*

}

Generics ograničenja

- *Holder<T>*
  - ako strogo definiramo *Holder<Integer>* - gubimo polimorfizam*!*

safeSet(ints, "Milica");
*private static void safeSet(**Holder<Integer> holder**, **Integer value**) {*
*	**holder.setValue(value); // compile time error*** *- Holder<Integer>* može primiti samo *Integer*
}

  - *Holder<T>* - samo potpuno isti tipovi - gubimo polimorfizam!

safeSet(ints, "Milica");
*private static void safeSet(**Holder<T> holder**, **T value**) {*
*	holder.setValue(value);*
}

PECS

- ***P**roducer **E**xtends **C**onsumer **S**uper*
- povratak fleksibilnosti *Array*-a
- ***Producer Extends*** – omogućava *covariance* – želimo iz *Holdera* pročitati *Number, ali* i sve njegove **podtipove** – mora biti tipiziran na *Number* ili njegov **podtip** (**extends**) da ga može pročitati

```
Number n = produceValue(new Holder<>(Math.PI));


private static Number produceValue(Holder<? extends Number> num) {
    Number value = num.getValue();
    return value;
}
```

PECS

- ***Consumer super*** – omogućava *contra-variance* – želimo u *Holder* upisati *Number* i sve njegove **podtipove** - mora biti tipiziran na *Number* ili njegov **nadtip** (**super**) da bi mogao upisati *Number*

```
Holder<Object> obj = new Holder<>();
consumeValue(obj, Math.PI);

private static void consumeValue(Holder<? super Number> num, Number value) {
    num.setValue(value);
}
```

Demo

- Project

Izvor:http://www.jnhsolutions.com/contact-us/request-a-demo/

Hvala na pažnji!