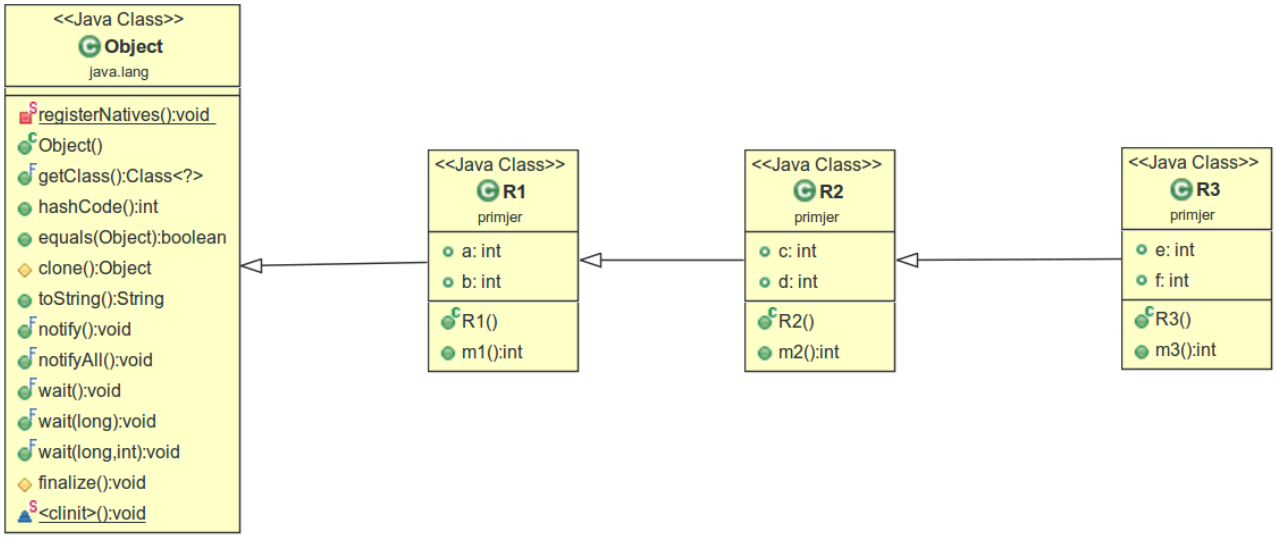
Tehnologija Java Generics: ograde na parametre u metodama

Pretpostavimo da imamo sljedeću hijerarhiju razreda.



Neka smo napravili kolekciju koja je definirana na sljedeći način:

```
Collection<R2> kolekcija = new ArrayIndexedCollection<>();
```

Razmislite što bismo od sljedećega htjeli da bude legalno?

```
kolekcija.add(new Object());       // da ili ne?
kolekcija.add(new R1());           // da ili ne?
kolekcija.add(new R2());           // da ili ne?
kolekcija.add(new R3());           // da ili ne?
```

OK, sada kada znamo koji se stvarno sve objekti mogu nalaziti u našoj promatranoj kolekciji, idemo dalje. Neka su definirani sljedeći procesori:

```
Processor<Object> p0 = new Processor<Object>() {
     @Override
     public void process(Object element) {
          System.out.println(element.hashCode());
     }
};

Processor<R1> p1 = new Processor<R1>() {
     @Override
     public void process(R1 element) {
          System.out.println(
               element.hashCode() + element.a
          );
     }
};
```

```
Processor<R2> p2 = new Processor<R2>() {
     @Override
     public void process(R2 element) {
          System.out.println(
               element.hashCode() + element.a + element.c
          );
     }
};

Processor<R3> p3 = new Processor<R3>() {
     @Override
     public void process(R3 element) {
          System.out.println(
               element.hashCode() + element.a +
               element.c + element.e
          );
     }
};
```

Primijetite što svaki od ovih procesora može dohvaćati iz objekata koje dobiva kao argument u metodi `process`.

Kako naša kolekcija nudi metodu `forEach` koja prima jedan procesor koji obrađuje sve objekte iz kolekcije (objekt po objekt), što bismo od sljedećega htjeli dopustiti, odnosno što ima smisla?

```
kolekcija.forEach(p0);        // da ili ne?
kolekcija.forEach(p1);        // da ili ne?
kolekcija.forEach(p2);        // da ili ne?
kolekcija.forEach(p3);        // da ili ne?
```

Razumijete li zašto? Koja je tada korektna deklaracija metode `forEach` u sučelju `Collection`? Koja je najopćenitija i korektna (pa u tom smislu ispravna)?

```
interface Collection<T> {
     void forEach(Processor<T> processor);
}

interface Collection<T> {
     void forEach(Processor<? extends T> processor);
}

interface Collection<T> {
     void forEach(Processor<? super T> processor);
}
```

Idemo dalje. Neka imamo sljedeće testere. Primijetite što svakom od njih stoji na raspolaganju od objekata koje primaju u metodi `test`.

```
Tester<Object> t0 = new Tester<Object>() {
     @Override
     public boolean test(Object element) {
          return element.hashCode() > 3;
     }
};

Tester<R1> t1 = new Tester<R1>() {
     @Override
     public boolean test(R1 element) {
          return element.hashCode() > 3
                    && element.a < 5;
     }
};

Tester<R2> t2 = new Tester<R2>() {
     @Override
     public boolean test(R2 element) {
          return element.hashCode() > 3
                    && element.a < 5
                    && element.c < 8;
     }
};

Tester<R3> t3 = new Tester<R3>() {
     @Override
     public boolean test(R3 element) {
          return element.hashCode() > 3
                    && element.a < 5
                    && element.c < 8
                    && element.d < 12;
     }
};
```

Sada razmatramo metodu `addAllSatisfying` koja prima jednu novu kolekciju i jedan tester, odnosno u sučelju `Collection<T>` ima neparametriziranu signaturu:

```
void addAllSatisfying(Collection other, Tester tester);
```

Ako smo “mi” kolekcija koja deklarirano pohranjuje elemente tipa `T`, koja je korektna parametrizacija argumenta `other`, odnosno kolekcija čega smije biti ta kolekcija da bismo mi smjeli u sebe dodati te elemente? Koja je najopćenitija i korektna (pa u tom smislu ispravna)?

```
Collection<T> other                    // da ili ne?
Collection<? extends T> other          // da ili ne?
Collection<? super T> other            // da ili ne?
```

Idemo dalje. Koje testere smijemo koristiti pri testiranju ako smo kolekcija elemenata tipa `T`? Ako je `kolekcija2` kolekcija prikladnog tipa određenog na temelju prethodnog razmatranja, što bismo htjeli koristiti za testiranje:

```
kolekcija.addAllSatisfying(kolekcija2, t0);  // da ili ne?
kolekcija.addAllSatisfying(kolekcija2, t1);  // da ili ne?
kolekcija.addAllSatisfying(kolekcija2, t2);  // da ili ne?
kolekcija.addAllSatisfying(kolekcija2, t3);  // da ili ne?
```

Na temelju toga, koja je korektna parametrizacija drugog argumenta ove metode? Koja je najopćenitija i korektna (pa u tom smislu ispravna)?

```
Tester<T>                // da ili ne?
Tester<? super T>        // da ili ne?
Tester<? extends T>      // da ili ne?
```

U konačnici, to znači da promatrana metoda treba imati kakvu signaturu?

```
void addAllSatisfying(
     Collection<_____________________> other,
     Tester<_____________________> tester
);
```