Budapesti Műszaki és Gazdaságtudományi Egyetem

Java technológia

Soft, weak és phantom referenciák

Bevezetés

- A Java speciális **referencia objektum**ai segítségével a **garbage collector** (szemétgyűjtő) működését (illetve objektumainkra gyakorolt hatását) befolyásolhatjuk. Ezeket az osztályokat a **java.lang.ref** csomagban találjuk.
- Ezen osztályok egy szokványos **Java referenciát burkolnak be**, funkciójuk nagyjából az, hogy leválasszák a referenciát a program többi részéről, vagyis úgy viselkednek, mintha a hivatkozott objektumot "kesztyűvel" fognánk csak meg, úgy, hogy a garbage collector "ne tudja meg".
- Ahhoz, hogy ezen burkolók hasznát megértsük, tekintsük át röviden a garbage collector működését.

A garbage collector működése

- A legtöbb, a dinamikus memóriakezelést ismerő, nyelvvel ellentétben a Java-ban objektumokat **csak létrehozni tudunk**, explicite törölni nem.
- **Az objektumokat** a virtuális gép egy speciális része, **a garbage collector törli ki**, amikor már "nincs rájuk szükség".
- Egy objektumra "nincs szükség", ha **nem mutat rá egy referencia sem** (nem elérhető), hiszen ilyenkor úgy sem tudunk már hozzáférni.
- Ebben az esetben a garbage collector az adott objektumot, és a csak általa hivatkozott egyéb objektumokat megszünteti, és az általuk lefogalt memóriát felszabadítja.
- Azon objektum-orientált nyelvekben, ahol van explicit törlés, az objektumoknak konstruktor mellett destruktoruk is van, ezt a Java-ban az objektum **finalize** metódusa helyettesíti, amelyet a garbage collector az objektum végleges törlése előtt hív meg.

Referencia-burkolók I.

- Előfordulhat, hogy egy objektumra hivatkozunk ugyan, de "nem ragaszkodunk hozzá". Ilyen esetekben beburkolhatjuk az objektumra mutató referenciát egy **referencia-burkoló** objektumba.
- Három típusú burkoló létezik, melyek mind a **Reference** ősosztályból származnak, ezek a **SoftReference**, a **WeakReference**, és a **PhantomReference**.
- A burkolóktól el lehet kérni az eredeti referenciát (referent), így az eredetileg hivatkozott objektumhoz továbbra is hozzáférünk.
- A burkolót **törölhetjük** a clear metódus segítségével, így a hivatkozott objektum már a burkolón keresztül sem hozzáférhető.
- A törlés kivételével a burkoló nem módosítható.
- Ha a burkoló elérhetetlenné válik, a burkolt objektum is megszűnhet.

Referencia-burkolók II.

- Lehetőségünk van arra, hogy értesítést kapjunk, amikor egy adott objektum már csak a burkolón keresztül hozzáférhető.
- E célból a burkolót regisztrálhatjuk egy **ReferenceQueue** objektumhoz.
- Amikor a garbage collector úgy találja, hogy az objektum hozzáférhetősége elérte az adott típusú burkoló szintjét (vagyis már csak azon keresztül érhető el az objektum), a burkolót **besorolja** abba a sorba, amelyhez regisztráltuk.
- A ReferenceQueue nem tud a hozzá regisztrált burkolókról, így a burkolót a programnak kell "életben tartania", amíg szükség van rá.
- A ReferenceQueue-tól lekérdezhetjük, van-e benne besorolt referencia-burkoló, és azokat kivehetjük belőle, további feldolgozás céljából.

Objektumok elérhetősége

- A következő elérhetőségi szintek rendelhetők egy objektumhoz:
  - **strongly reachable** (erősen elérhető): normál (erős) referenciákon keresztül elérhető,
  - **softly reachable** ("lágyan" elérhető): nem erősen elérhető, csak SoftReference példányokon keresztül érhető el,
  - **weakly reachable** (gyengén elérhető): nem erősen, vagy "lágyan" elérhető, csak WeakReference példányokon keresztül érhető el,
  - **phantom reachable** ("fantom" elérhető): nem erősen, "lágyan", vagy gyengén elérhető, az objektum finalize metódusa már lefutott, csak PhantomReference példányokon keresztül érhető el,
  - **unreachable** (nem elérhető): az objektum nem elérhető, az általa használt memóriaterület felszabadítható.

A burkolók ereje

- A burkolók az egyre gyengébb elérhetőségi szinteknek felelnek meg, és eltérő tulajdonságokkal bírnak:
  - **SoftReference**: Ha egy objektum "lágyan" elérhető, és a garbage collector "megszorul", vagyis memóriahiány miatt nem tud kiszolgálni memóriaigényeket, lehetősége van az objektumra mutató SoftReference példányok atomi törlésére, ekkor, azon SoftReference-ek, amelyeket regisztráltunk egy sorba (ReferenceQueue), automatikusan besorolásra kerülnek.
  - **WeakReference**: Ha egy objektum gyengén elérhető, a WeakReference példányok atomian törlődnek, és a regisztrált WeakReference-ek besorolásra kerülnek.
  - **PhantomReference**: Ha egy objektum fantom elérhető, a PhantomReference-ek besorolásra kerülnek, de nem törlődnek.

A SoftReference

- A SoftReference addig "ragaszkodik" az általa burkolt objektumhoz, amíg a garbage collector-nak nincs szüksége **extra memóriára valamely igény kielégítéséhez**.
- Ez a viselkedés kiválóan alkalmassá teszi **memória-érzékeny cache tárolók** kialakítására.
- Intelligensebb cache tárolók megtehetik, hogy a gyakran használt elemeket erős referenciákkal rögzítik, így azok SoftReference burkolóit a garbage collector nem törli (mert erősen elérhetőek).
- A garbage collector nem köteles először a régebben használt SoftReference-eket törölni, de ez a viselkedés javasolt az garbage collector implementációk számára.

A WeakReference

- A WeakReference gyengébb a SoftReference-nél: egy gyengén elérhető objektum **azonnal törölhető**, nem csak memóriahiány esetén.
- Ezeket a burkolókat például **kanonizáló leképezések** készítésére használhatjuk (olyan leképezések, amelyek egy egyedi azonosítót egy objektumra képeznek le).
- Feltételezzük, hogy olyan nagyméretű objektumokról van szó, amelyeket bármikor betölthetünk (például adatbázisból), de nem tarthatjuk őket egyszerre a memóriában.
- Ha egy leképezésben az értékek helyére WeakReference objektumokat helyezünk, akkor a nem használt értékek automatikusan törlődnek. (Azok, amelyekre már nem mutat erős referencia.)
- A burkolót felhasználás előtt megvizsgáljuk, ha törlődött, az értéket újra betöltjük.

A PhantomReference

- A PhantomReference némileg másképpen működik, mint a másik két referencia-burkoló.
- Ha egy objektum fantom elérhetővé válik, a PhantomReference **nem törlődik automatikusan**, így csak akkor használható értelmesen, ha egy ReferenceQueue-hoz regisztráljuk (ezt kötelező is megtenni).
- **A get metódussal nem kaphatjuk vissza a referencia értékét** (a get mindig null-t ad), mert egy fantom elérhető objektum már gyakorlatilag megszűnt (mert a finalize metódusa lefutott), így nem engedhető meg, hogy elérhetőségi állapota megváltozzon.
- A PhantomReference-t **manuálisan kell törölni**!
- A PhantomReference segítségével szinkronizált tisztogatást végezhetünk, például egy erőforrás felszabadítása, ha már az összes felhasználója megszűnt.

A ReferenceQueue

- A ReferenceQueue a garbage collector (vagy a program) által besorolt referencia objektumokat tartalmazza.
- A **poll** metódussal lekérdezhetjük, hogy van-e besorolt referencia. Ha van, visszakapjuk az elsőt, ha nincs, null-t kapunk.
- A **remove** metódussal várhatunk egy referencia besorolására. Ha azonnal van besorolt referencia, a metódus visszatér, ha nincs, blokkolódik, amíg egy referencia be nem kerül a sorba, vagy a megadott timeout le nem jár.
- A garbage collector a PhantomReference-ek kivételével törli a besorolt referenciákat besorolás előtt, így a poll, illetve remove metódus által visszaadott referencia objektumból a hivatkozott objektumot már nem nyerhetjük ki.

A WeakHashMap

- A java.util.WeakHashMap osztály egy olyan **hash leképezés**t valósít meg, ahol a kulcsok WeakReference-eken keresztül tárolódnak.
- Az, hogy egy objektum egy WeakHashMap kulcsaként használt, nem akadályozza meg a garbage collector-t abban, hogy az objektumot felszámolja.
- Ha egy kulcsként használt objektum megszűnik, **a hozzá tartozó leképezés automatikusan törlődik** a WeakHashMap-ből.
- Mivel a garbage collector a háttérben működik, a WeakHashMap-ben lévő leképezések száma bármikor megváltozhat.

Alkalmazási példa I.

- Feladat: készítsünk egy eseményeket generáló osztályt, amelyhez regisztrálhatják magukat az eseményekben érdekelt objektumok. Az eseménygeneráló osztályhoz való regisztráció ne akadályozza az osztályok felszabadítását.
- Megoldás: WeakReference-eket tartalmazó lista használata a listener-ek tárolására.
- Első lépés: az esemény objektum, és a listener interfész elkészítése.

```
public class Event { ... }

public interface Listener {
    void eventOccurred(Event e);
}
```

- Második lépés: az eseménygeneráló osztály elkészítése

Alkalmazási példa II.

```
import java.lang.ref.*;
import java.util.*;

public class Test {
   public Test () {
      queue = new ReferenceQueue (); listeners = new ArrayList ();
   }
   public void registerListener (Listener l) {
      listeners.add (new WeakReference (l,queue));
   }
   public void fireEvent (Event e) {
      Reference ref;
      while ((ref = queue.poll ()) != null) listeners.remove (ref);
      Iterator i = listeners.iterator ();
      while (i.hasNext ()) {
         Listener l = (Listener) ((Reference) i.next ()).get ();
         if (l != null) l.eventOccurred (e);
      }
   }
   private ReferenceQueue queue;
   private ArrayList      listeners;
}
```