# Concurrence et parallélisme

Architecture logicielle

Première partie

# Concurrence != Parallélisme

- Concurrence: Gérer beaucoup de choses en même temps.
  - Gérer un grand nombre d’opérations ou demandes sur une même ressource.
- Parallélisme: Faire beaucoup de choses en même temps.
  - Découper un problème en sous-parties qui sont faisables en même temps et regrouper le résultat à la fin.

# Exercice sur la concurrence

- Problème
  - La base de données dans votre projet de session se fait trop accéder, le nombre de connexions réseau et la latence associée aux recherches pourraient être optimisés.
- Solution
  - Implémentation d’une cache pour optimiser les requêtes faites à la base de données.
- Code du laboratoire
  - https://github.com/bgagnonadam/concurrency-testing

# Exercice sur la concurrence

- MultithreadedTC
  - https://www.cs.umd.edu/projects/PL/multithreadedtc/overview.html
  - S'intègre facilement à une suite de tests JUnit
- Trois phases à un test MtTC
  - initialize()
    - équivalent d'un @Before
  - thread#()
    - code de chaque thread à exécuter
  - finish()
    - les asserts qui sont dans le "//then" d'un test
- Synchronizer vos thread à un endroit
  - waitForTick(1);

```
class MTCBoundedBufferTest extends MultithreadedTestCase {
    ArrayBlockingQueue<Integer> buf;
    @Override public void initialize() {
        buf = new ArrayBlockingQueue<Integer>(1);
    }

    public void thread1() throws InterruptedException {
        buf.put(42);
        buf.put(17);
        assertTick(1);
    }

    public void thread2() throws InterruptedException {
        waitForTick(1);
        assertEquals(Integer.valueOf(42), buf.take());
        assertEquals(Integer.valueOf(17), buf.take());
    }

    @Override public void finish() {
        assertTrue(buf.isEmpty());
    }
}
```

# Exercice sur la concurrence

- Exécuter votre test MtTC

```
public void testMTCBoundedBuffer() throws Throwable {
    TestFramework.runOnce( new MTCBoundedBufferTest() );
}
```

- Dans notre problème, nous voulons voir si la base de données sera appelée plus d'une fois lorsqu'on essaie d'obtenir les informations de la même résidence plus d'une fois en simultané

```
public RealEstate getRealEstate(String id) {
    RealEstate realEstate = realEstates.get(id);
    if (realEstate == null) {
        realEstate = realEstateRepository.findById(id);
        realEstates.put(id, realEstate);
    }
    return realEstate;
}
```

# Exercice sur la concurrence

- Quelques minutes pour essayer notre cache avec MultithreadedTC.

# Exercice sur la concurrence

- Le test MtTC ne passe pas…
- Solution
  - Double Check Lock!
- Attention au JIT si votre condition est déterministe
  - http://www.javaworld.com/article/2074979/java-concurrency/double-checked-locking--clever--but-broken.html

```
class SomeClass {
  private Resource resource = null;
  public Resource getResource() {
    if (resource == null) {
      synchronized {
        if (resource == null)
          resource = new Resource();
      }
    }
    return resource;
  }
}
```

# Exercice sur la concurrence

- Quelques minutes pour essayer notre cache avec un DCL

- Un exemple un peu plus complet avec une purge de la cache par un autre thread périodiquement.

# Exercice sur la concurrence

- Mais la question qui tue: Quelles seraient les alternatives au Double Check Lock?

# Examen la semaine prochaine!

- Questions ?