SpotBugs

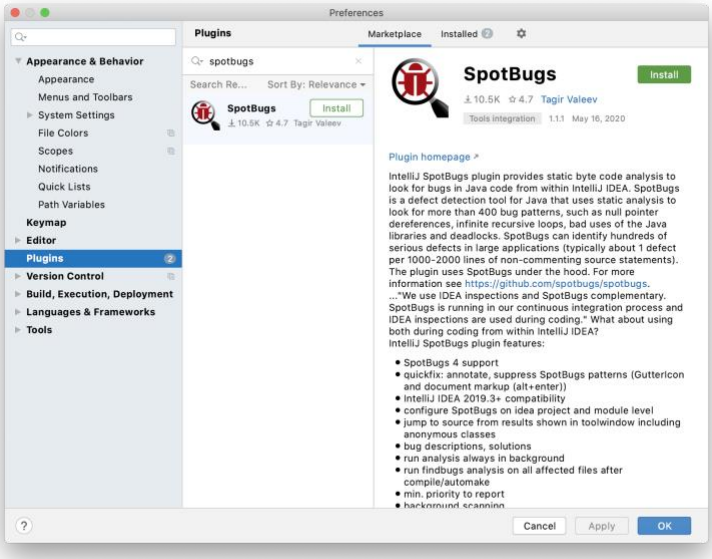
In diesem Arbeitsblatt suchen Sie nach Concurrency Problemen in Java Code. Im ersten Anlauf verwenden Sie Ihren Verstand und danach SpotBugs [1] um die Probleme zu lokalisieren.

Aufgaben:

a) Finden Sie die Concurrency Probleme in den Klassen auf der Rückseite.

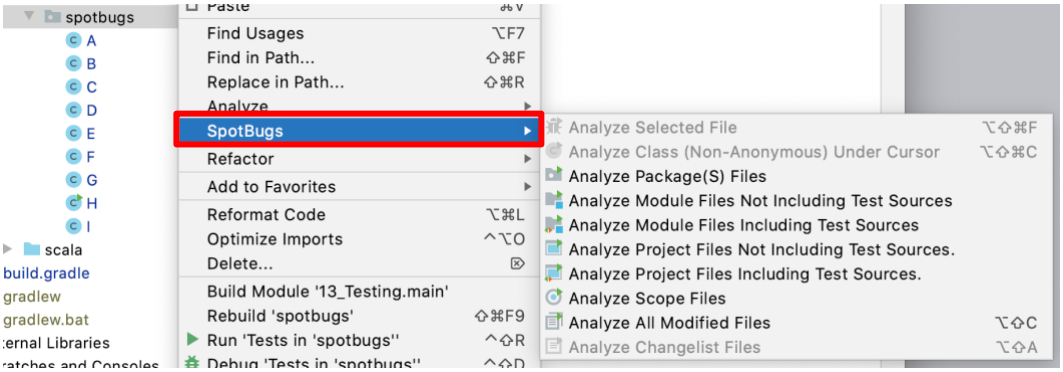
b) Installieren Sie SpotBugs in Ihre IntelliJ Umgebung:

Preferences -> Plugins und suchen Sie im Marketplace nach "spotbugs"



Wählen Sie SpotBugs, klicken Sie "Install" und starten Sie die IntelliJ neu.

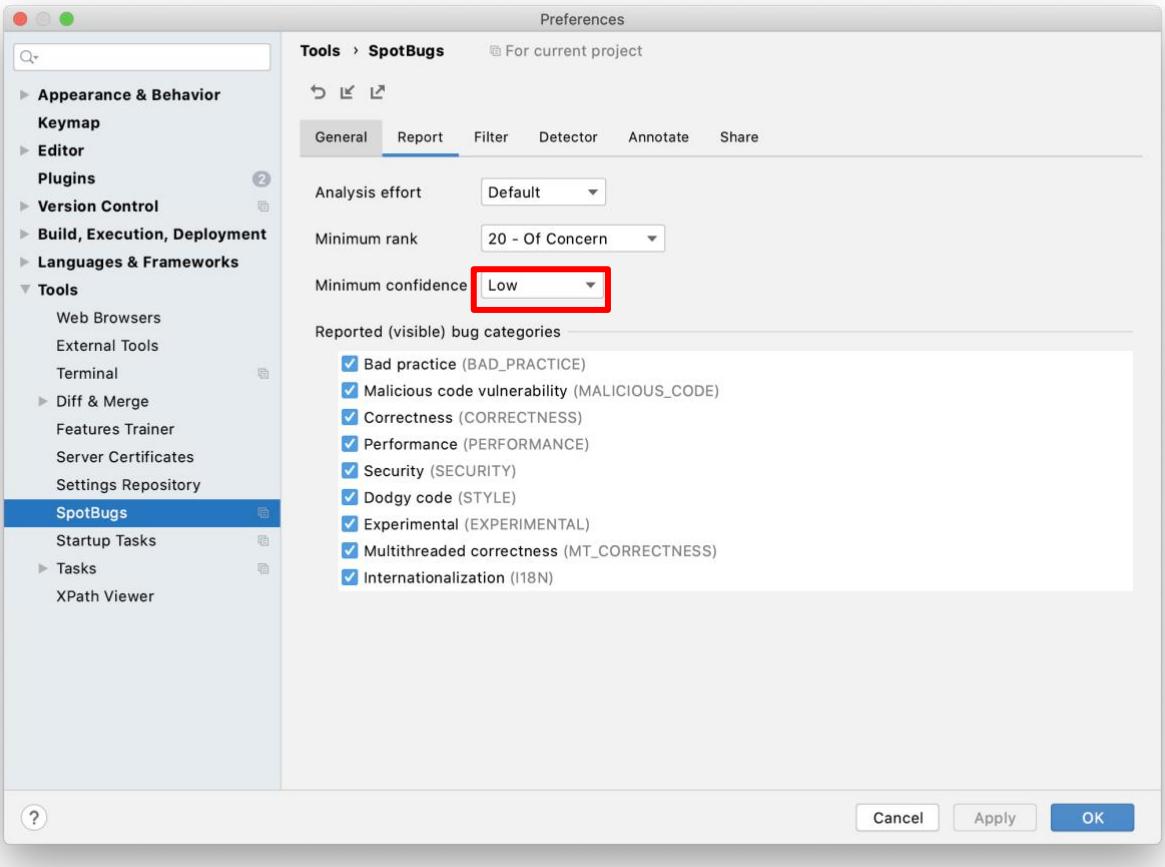
c) Verwenden Sie SpotBugs um die Probleme automatisch zu lokalisieren. Im Kontextmenü auf dem Projekt/Package/Klasse aktivieren Sie die Spot Bugs Actions. Die problematischen Stellen werden mit einem Bug markiert.



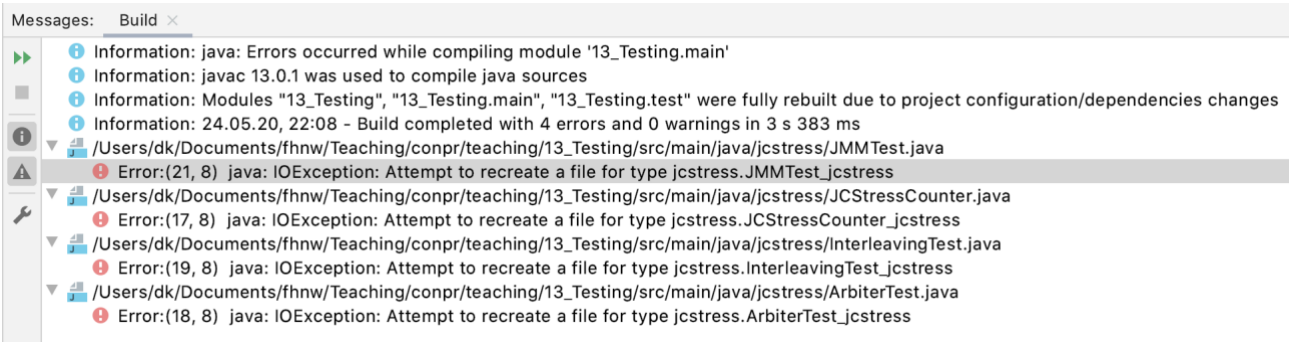
[1] SpotBugs Home: https://spotbugs.github.io/
[2] Auch für Eclipse verfügbar: https://marketplace.eclipse.org/content/spotbugs-eclipse-plugin

Hinweise:

Damit SpotBugs alle Fehler findet, müssen Sie in den Preferences die "Minimum confidence" auf Low stellen:



Wenn folgende Build Fehler beim Ausführen von Spotbugs auftreten:



hilft ein `gradle clean`.

```java
public class A {
  private final Lock lock
    = new ReentrantLock();

  private int id;

  public int nextId() {
    synchronized(lock) {
      return id++;
    }
  }
}
```

```java
public class B {
  private final String lock = "LOCK";

  private int id;

  public int nextId() {
    synchronized(lock) {
      return id++;
    }
  }
}
```

```java
public class C {
  private static C instance;

  public static C instance() {
    if(instance == null) {
      synchronized (C.class) {
        if(instance == null) {
          instance = new C();
        }
      }
    }
    return instance;
  }
}
```

```java
public class D {
  @GuardedBy("this")
  private int i;

  public synchronized void inc() {
    i++;
  }

  public int get() {
    return i;
  }
}
```

```java
public class E {
  private Long sequence = Long.valueOf(0);

  public Long next() {
    synchronized (sequence) {
      sequence = sequence.longValue() + 1;
      return sequence;
    }
  }
}
```

```java
public class F {
  private final Lock lock = new ReentrantLock();
  private final Condition cond = lock.newCondition();

  private Object elem;

  public Object get() throws InterruptedException {
    lock.lock();
    try {
      while(elem == null) {
        cond.wait();
      }
      return elem;
    } finally {
      lock.unlock();
    }
  }

  public void put(Object elem) {
    lock.lock();
    try {
      this.elem = elem;
      cond.notifyAll();
    } finally {
      lock.unlock();
    }
  }
}
```

```java
public class G {
  private Object elem;

  public synchronized Object get()
       throws InterruptedException {
    if (elem == null) { wait(); }
    return elem;
  }

  public void put(Object elem) {
    this.elem = elem;
    notify();
  }
}
```

```java
public class H {
  public static void main(String[] args) {
    new Thread(new Runnable() {
      public void run() {
        System.out.print(Thread.currentThread());
      }
    }).run();
  }
}
```

```java
public class I extends LinkedList<Integer> {
  public I() {
    new Thread(new Runnable() {
      public void run() {
        add(1);
      }
    }).start();
  }
}
```