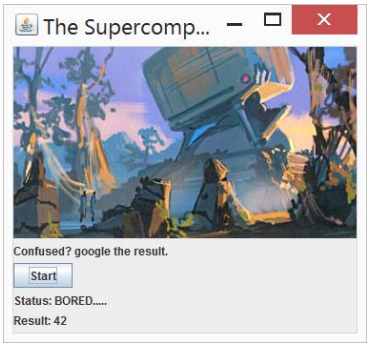
Übung 7: GUI und Threading

Zielsetzung:

- Single-Thread GUI Modell verstehen und nachvollziehen.
- Nicht-blockierende GUIs mit Thread Auslagerung und Dispatching realisieren.
- GUI und Threading für Swing und .NET WPF anwenden.
- Das C# async/await Programmiermodell untersuchen und einsetzen.

Aufgabe 1: Reaktionsfähiges Swing GUI

Die Vorlage enthält ein lauffähiges Java Swing UI, welches jedoch durch die lange Rechenzeit der Klasse `TheSupercomputer` beim Klicken auf „Start“ für längere Zeit blockiert wird.



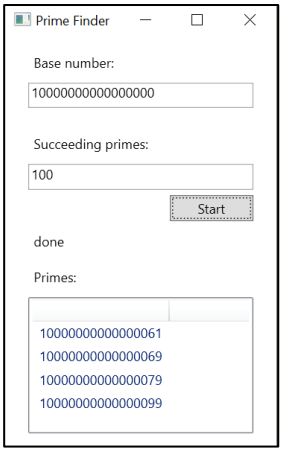
a) Finden Sie das Problem. Welcher Methoden-Aufruf verursacht das Blockieren?

b) Ändern Sie den Code, so dass es nicht blockiert.

c) Erreichen Sie, dass der Status des Supercomputers während dem Laufen angezeigt wird.

Aufgabe 2: Nicht-blockierendes WPF GUI

Analog zur Aufgabe 1 finden Sie in der Vorlage eine einfache .NET WPF Anwendung. Diese erlaubt es, ausgehend von einer Basiszahl die nächsten nachfolgenden Primzahlen zu finden. Leider blockiert auch dieses GUI während der Rechnung.



a) Wenden Sie das async/await Programmiermodell an, um das GUI reaktionsfähig zu machen.

b) Erweitern Sie die Anwendung um folgende Features:

- Fortschrittsanzeige (Progress Information) während der Rechnung, z.B. „20% computed“. Der Fortschritt muss nicht exakt mit der Gesamtzeit korrelieren.

- Während der Rechnung soll der Button „Cancel“ heissen, so dass man eine laufende Rechnung damit abrechen kann.

Aufgabe 3: async/await Lab (fakultativ)

Untersuchen und debuggen Sie eine Reihe von kleinen C#-Anwendungen, die alle das async/await Modell einsetzen. Versuchen Sie jeweils den Grund des Verhaltens zu verstehen:

- **Pitfall 1**: Versehentlich blockierender async Aufruf
- **Pitfall 2**: Thread-Wechsel innerhalb derselben Methodenausführung
- **Pitfall 3**: Pseudo-Concurrency: Drücken Sie „Start Download“ und dann schnell den „Add“ Knopf.
- **Pitfall 4**: Race Condition in einer asynchronen Anwendung
- **Pitfall 5**: Plötzlicher Deadlock im GUI