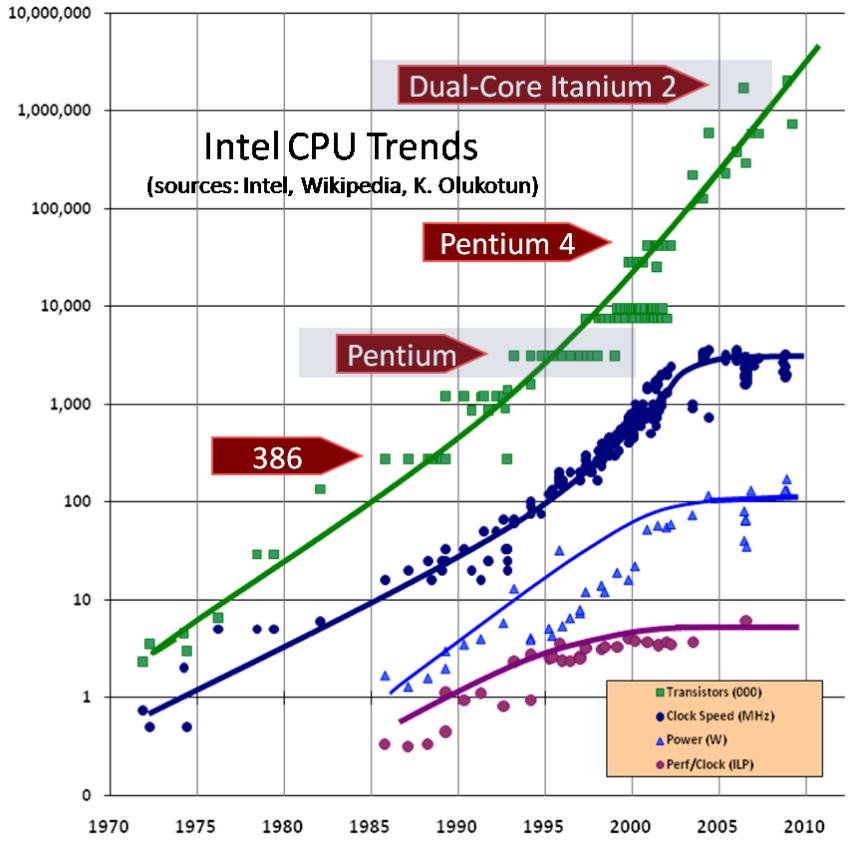
Parallelität in grafischen Oberflächen

Parallelprogrammierung mit JavaFX

Hochschule **Rosenheim**
University of Applied Sciences

„The free lunch is over“ – Es gibt keine kostenlose Performanzsteigerung in der Multi-Core-Era



Seit 2004 ist die Taktfrequenz von CPUs konstant

Hochschule Rosenheim
University of Applied Sciences

Was hat das alles mit GUI zu tun?

- Alle Mainstream UI-Bibliotheken sind NICHT-Threadsafe
- Lange Aktionen dürfen nicht vom Haupt-UI-Thread (FX-Applikations-Thread) ausgeführt werden – sonst blockiert alles
- Parallelität ist der Normalfall – nicht der Sonderfall

Hochschule **Rosenheim**
University of Applied Sciences

Wie greift man aus einem parallelen Thread richtig auf die Oberfläche zu?

- Alle Zugriffe auf UI-Elemente aus parallelen Threads müssen in ein Runnable verpackt werden
- Das Kommando Platform.runLater führt die folgende Methode aus dem FX-Applikationsthread aus

```
Platform.runLater(new Runnable() {
      @Override
      public void run() {
          // Update the text node with calculated results
      }
 });
```

Hochschule Rosenheim
University of Applied Sciences

Das Paket javafx.concurrent

- Vereinfacht die Parallelprogrammierung
- Standardprobleme an der UI
  - Wie kann man eine Aktion abbrechen?
  - Fortschrittsbalken
  - Nachrichten bei länger dauernden Aktionen

Hochschule Rosenheim
University of Applied Sciences

Die Worker Schnittstelle

-
- * A Worker is an object which performs some work in one or more background
- * threads, and who's state is observable and available to JavaFX applications
- * and is usable from the main JavaFX Application thread. This interface is
- * primarily implemented by both {@link Task} and {@link Service}, providing
- * a common API among both classes which makes it easier for libraries and
- * frameworks to provide workers which work well when developing user interfaces.
- *

Hochschule Rosenheim
University of Applied Sciences

Die Klasse Task

- Implementiert das Worker Interface
- Änderungen an den Properties werden aus dem FX-Applikationsthread durchgeführt -> safe to bind
- Ist abstrakt
  - Der Nutzer muss ableiten und die call() Methode überschreiben
    - Template Method Pattern
- Die call() Methode enthält die parallele Logik und kann einen optionalen Rückgabewert haben
  - Die Methoden succeed(), failed() und canceled() können überschrieben werden – Diese werden aus dem Applikationsthread aufgerufen. Die Methode succeed() kann per getValue() auf den Rückgabewert der call() Methode zugreifen

Hochschule **Rosenheim**
University of Applied Sciences

Wie startet man einen Task?

- Ein Task ist ein “Runnable” und kann einem Thread im Konstruktor übergeben werden
  - `Thread th = new Thread(task);`
  - `th.setDaemon(true);`
  - `th.start();`

- Ein Task kann per java.util.concurrent.ExecutorService gestartet werden
  - `ExecutorService executor = Executors.new`*`XYExecutor(…);`*
  - `executor.submit(task);`

Hochschule **Rosenheim**
University of Applied Sciences

Tasks müssen sich selbst abbrechen

Example 1

```
import javafx.concurrent.Task;

Task<Integer> task = new Task<Integer>() {
    @Override protected Integer call() throws Exception {
        int iterations;
        for (iterations = 0; iterations < 100000; iterations++) {
            if (isCancelled()) {
                break;
            }
            System.out.println("Iteration " + iterations);
        }
        return iterations;
    }
};
```

Rosenheim
pplied Sciences

Viele Methoden (IO/Sleep) werden beim Abbruch (cancel) durch eine InterruptedException unterbrochen

Example 2

```
import javafx.concurrent.Task;

Task<Integer> task = new Task<Integer>() {
    @Override protected Integer call() throws Exception {
        int iterations;
        for (iterations = 0; iterations < 1000; iterations++) {
            if (isCancelled()) {
                updateMessage("Cancelled");
                break;
            }
            updateMessage("Iteration " + iterations);
            updateProgress(iterations, 1000);

            //Block the thread for a short time, but be sure
            //to check the InterruptedException for cancellation
            try {
                Thread.sleep(100);
            } catch (InterruptedException interrupted) {
                if (isCancelled()) {
                    updateMessage("Cancelled");
                    break;
                }
            }
        }
        return iterations;
    }
};
```

Hochschule **Rosenheim**
University of Applied Sciences

Ein vollständiger Test

```
public class App extends Application {

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) throws Exception {

        Task<String> task = new Task<String>() {
            @Override
            protected String call() throws Exception {
                System.out.println(Thread.currentThread());
                return "Hugo";
            }

            @Override
            protected void succeeded() {
                System.out.println(Thread.currentThread() + " : " + super.getValue());
            }
        };
        new Thread(task).start();
    }
}
```

Hochschule Rosenheim
University of Applied Sciences

Die Properties eines Tasks können per Binding mit der UI verbunden werden

Example 3

```
import javafx.concurrent.Task;

Task task = new Task<Void>() {
    @Override public Void call() {
        static final int max = 1000000;
        for (int i=1; i<=max; i++) {
            if (isCancelled()) {
               break;
            }
            updateProgress(i, max);
        }
        return null;
    }
};
ProgressBar bar = new ProgressBar();
bar.progressProperty().bind(task.progressProperty());
new Thread(task).start();
```

Die Klasse Service

- Service ist eine Hilfklasse zur Nutzung von Tasks aus dem JavaFX Applikationsthread.
- Kapselt den Thread-Pool- / Executor-Code
- Die Klasse ist abstrakt. Der Nutzer muss die Methode createTask() überschreiben. Diese Methode erzeugt den Task.
- Services können direkt per start() gestartet werden
- Mit restart() kann der Task abgebrochen und neu gestartet werden
- Die Klasse Service implementiert die Worker-Schnittstelle
- Das Service-Objekt wird i.R. pro Controller 1x erzeugt

Hochschule **Rosenheim**
University of Applied Sciences

Ein vollständiges Beispiel

Nutzung

```
public class FirstLineServiceApp extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        FirstLineService service = new FirstLineService();
        service.setUrl("http://google.com");
        service.setOnSucceeded(new EventHandler<WorkerStateEvent>() {

            @Override
            public void handle(WorkerStateEvent t) {
                System.out.println("done:" + t.getSource().getValue());
            }
        });
        service.start();
    }

    public static void main(String[] args) {
        launch();
    }
```

```
private static class FirstLineService extends Service<String> {
    private StringProperty url = new SimpleStringProperty();

    public final void setUrl(String value) {
        url.set(value);
    }

    public final String getUrl() {
        return url.get();
    }

    public final StringProperty urlProperty() {
       return url;
    }

    protected Task<String> createTask() {
        final String _url = getUrl();
        return new Task<String>() {
            protected String call()
                throws IOException, MalformedURLException {
                    String result = null;
                    BufferedReader in = null;
                    try {
                        URL u = new URL(_url);
                        in = new BufferedReader(
                            new InputStreamReader(u.openStream()));
                        result = in.readLine();
                    } finally {
                        if (in != null) {
                            in.close();
                        }
                    }
                    return result;
                }
            };
    }
}
```