

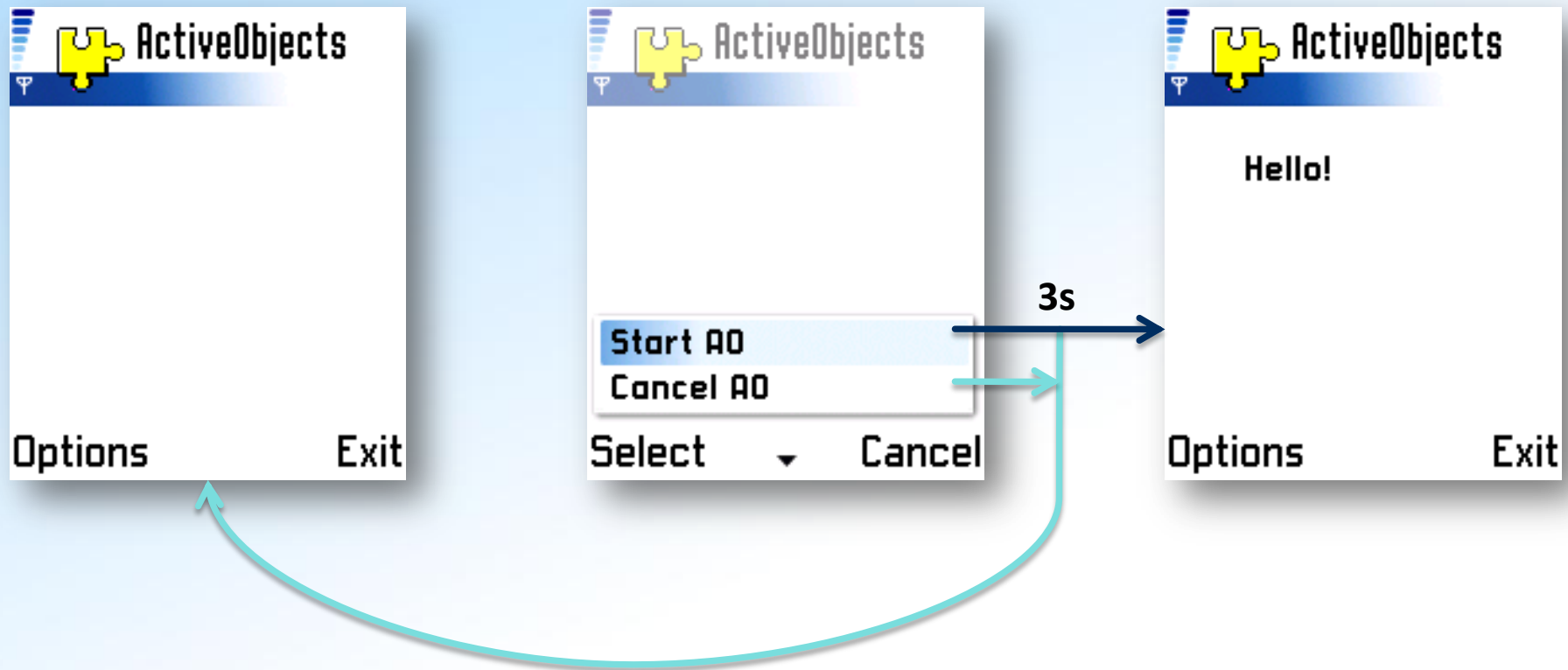
# **Symbian OS**

Challenge  
(Active Objects)

# Disclaimer

- These slides are provided free of charge at <http://www.symbianresources.com> and are used during Symbian OS courses at the University of Applied Sciences in Hagenberg, Austria ( <http://www.fh-hagenberg.at/> )
- Respecting the copyright laws, you are allowed to use them:
  - for your own, personal, non-commercial use
  - in the academic environment
- In all other cases (e.g. for commercial training), please contact [andreas.jakl@fh-hagenberg.at](mailto:andreas.jakl@fh-hagenberg.at)
- The correctness of the contents of these materials cannot be guaranteed. Andreas Jakl is not liable for incorrect information or damage that may arise from using the materials.
- Parts of these materials are based on information from Symbian Press-books published by John Wiley & Sons, Ltd. This document contains copyright materials which are proprietary to Symbian, UIQ, Nokia and SonyEricsson. “S60™” is a trademark of Nokia. “UIQ™” is a trademark of UIQ Technology. Pictures of mobile phones or applications are copyright their respective manufacturers / developers. “Symbian™”, “Symbian OS™” and all other Symbian-based marks and logos are trademarks of Symbian Software Limited and are used under license. © Symbian Software Limited 2006.

# The Application



# Strategy

- Design a simple UI using the UI-designer
- Create an own class for the **Active Object**
- Use `RTimer` for the call-back
  - See the **SDK-help** for details on `RTimer`!
- Request a **window redraw** in the container after updating the label
  - See the SDK-help for the container's base class **CCoeControl**
- **Warning:** Do not write anything into automatically managed code parts

# Code

- Set label text in the container that owns the label:

```
void CActiveObjectsContainer::SetLabelTextL ( const TDesC& aText )
{
    iLabelText->SetTextL(aText);
    DrawDeferred();           // Request a redraw
}
```

# Active Object – Declarations

```
#include <e32base.h>
#include <e32std.h>
#include "ActiveObjectsContainer.h"
class CDelayedProcess : public CActive { // Derive from CActive
public:
    // Construct / destruct – Symbian OS Two-phased construction
    static CDelayedProcess* NewL(CActiveObjectsContainer* aContainer);
    ~CDelayedProcess();
    // Send out a new request
    void StartProcess(TTimeIntervalMicroSeconds32 aDelay);
private:
    // Construct / destruct
    CDelayedProcess(CActiveObjectsContainer* aContainer);
    void ConstructL();
    void RunL(); // From CActive – asynchronous event handling
    void DoCancel(); // Internal cancel function
private:
    RTimer iTimer; // Asynch. service provider
    CActiveObjectsContainer* iContainer; // To change the label text
};
```

# Active Object – Construction

```
CDelayedProcess::CDelayedProcess(CActiveObjectsContainer* aContainer)
: CActive(CActive::EPriorityStandard),
iContainer(aContainer) {
    // Call base constructor of CActive with the standard priority
    // Add this active object instance to the Active Scheduler:
    CActiveScheduler::Add(this);
}

void CDelayedProcess::ConstructL() {
    // Create a thread-relative timer.
    User::LeaveIfError(iTimer.CreateLocal());
}

CDelayedProcess::~~CDelayedProcess() {
    // Call cancel-function of CActive base class, which will call DoCancel()
    Cancel();
    // Close the handle to the timer
    iTimer.Close();
}
```

# Active Object – Asynchronous

```
void CDelayedProcess::StartProcess(TTimeIntervalMicroSeconds32 aDelay) {
    // Make sure that no other request is currently active
    _LIT(KDelayedProcessPanic, "CDelayedProcess");
    __ASSERT_ALWAYS(!IsActive(), User::Panic(KDelayedProcessPanic, 1));

    iTimer.After(iStatus, aDelay);           // Request the callback from the ASP
    SetActive();                             // Mark our Active Object as active
}

void CDelayedProcess::RunL() {
    _LIT(KTxt, "Hello!");
    iContainer->SetLabelTextL(KTxt);        // Print text to the screen after the delay has passed
}

void CDelayedProcess::DoCancel() {
    iTimer.Cancel();                        // Stop the asynch. request
}
```

# Active Object – Call

```
void CActiveObjectsContainerView::DoActivateL(...) {           // The container has to exist already
    [...]
    if (!iDelayedProcess && iActiveObjectsContainer) {
        // Create a new instance of our Active Object class
        iDelayedProcess = CDelayedProcess::NewL(iActiveObjectsContainer);
    }
}
CActiveObjectsContainerView::~CActiveObjectsContainerView() {
    delete iDelayedProcess;           // Cleanup the AO – the destructor will call Cancel()
    [...]
}
TBool CActiveObjectsContainerView::HandleStart_AOMenuItemSelectedL( TInt aCommand ) {
    if (iDelayedProcess)           // Request the asynch. event in 3 seconds
        iDelayedProcess->StartProcess(3000000);
    return ETrue;
}
TBool CActiveObjectsContainerView::HandleCancel_AOMenuItemSelectedL( TInt aCommand ) {
    if (iDelayedProcess)           // Cancel the asynch. process
        iDelayedProcess->Cancel();
    return ETrue;
}
```