

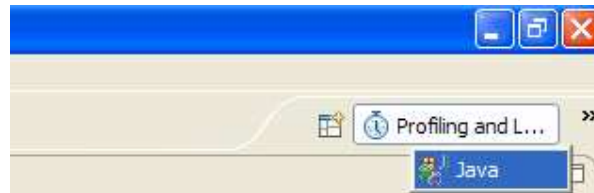
# Getting Started

## Install TPTP and samples

1. Please get TPTPDemo.zip file from instructors.
2. Unzip TPTPDemo.zip. Open TPTPDemo folder and there's a zip file Eclipse-TPTP-4.5.2.zip there.
3. Unzip Eclipse-TPTP-4.5.2.zip to current folder. Open eclipse subdirectory and Click eclipse.exe to start Eclipse.
4. Find DemoWorkSpace subdirectory of TPTPDemo. Input DemoWorkSpace directory in Workspace Launcher. Press OK to continue.

## Open perspective

1. Open Profiling and Logging perspective: choose menu "Window" -> "Open Perspective"->"Other..."-> Profiling and Logging.
2. Switch back to Java perspective: click Java in the top-right of workbench.



# Demo1: ProductSample

## Introduction:

ProductSample is a java application. It loads product data stored in XML format and parse the data to print it in console.

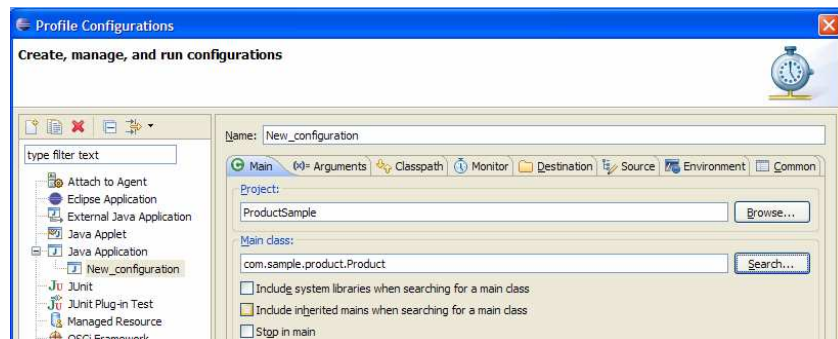
## Details:

You can open src directory of ProductSample project. There're two files there. Product.java contains main function. It will call readCatalogFromFolder function to read data and print data. Functions in ProductCatalog.java are responsible for reading data in and parse data.

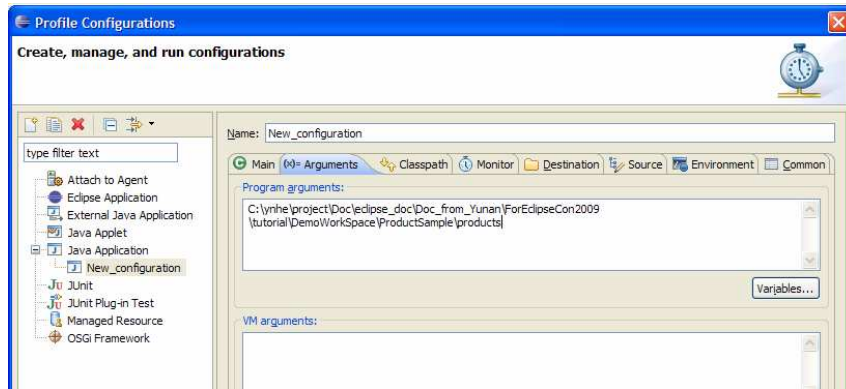
Data is stored in "products" subdirectory of ProductSample project.

## Steps to run:

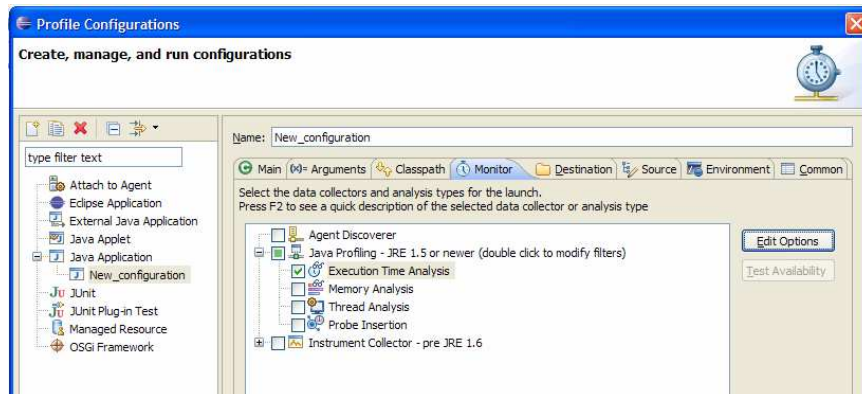
1. Open "Profile Configurations" Dialog. Double click "Java Application" in Monitor and Navigator view. Set "Project" to ProductSample and "Main class" to com.sample.product.Product.



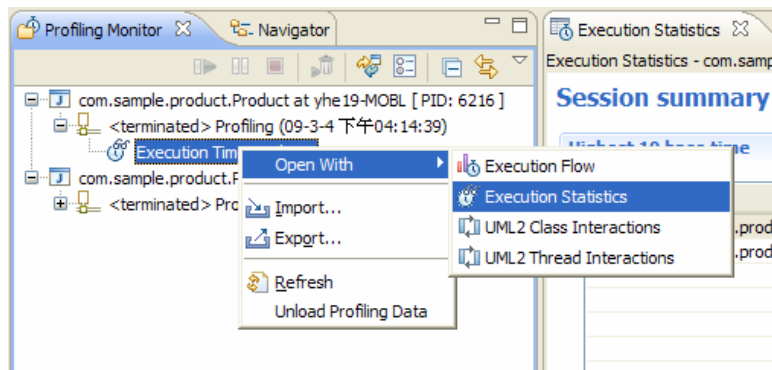
2. Open "Arguments" tab and input the directory of "products" (mentioned before).




3. Open "Monitor" tab and choose "Execution Time Analysis".



4. Click "Profile" button to profile.
5. Open "Profiling and Logging perspective" and view data with "Execution Statistics View"



## Steps to identify problem:

1. View statistics in Method level by clicking  in top-right of workbench.



2. Sort method by clicking <BaseTime> column header.

Package	<Base Time (s...	Average Base ..
com.sample....	1.745522	0.0727
com.sample....	0.203845	0.0084
com.sample....	0.042406	0.0424
com.sample....	0.001704	0.0000

- The hottest method is “createParser()”. Double click this method. It will navigate to “Method Invocation Details ” tab. You will found that this method is invoked by parseContent(java.io.File)void.

Execution Statistics - com.sample.product.Product at yhe19-MOBL [PID: 6216]

**Method Invocation Details**

Selected method

>Calls	Method	Class	Package	Base Time (sec...	Average Base ...	Cumulative Tim...	Cumula
24	createParser() javax.xml.parsers.SA...	ProductCatalog	com.sample.p...	1.745522	0.072730	1.745522	

Selected method is invoked by:

>Invoked by	Method	Class	Package	Base Time (sec...	Average Base ...	Cumulative Tim...	Cal
24	parseContent(java.io.File) void	ProductCatalog	com.sample.p...	0.203845	0.008494	1.950570	

- Right click on function “parseContent” and choose “Open Source ” in popup menu.

Selected method is invoked by:

Invoked by	<Method	Class	Package	Base Time (sec...	Avera
24	parseContent(java.io.File) void	ProductCatalog	com.sample.p...	0.203845	

Selected method invokes:

>Invokes	Method	Class	Package	Base Time (sec...	Avera
----------	--------	-------	---------	-------------------	-------

- Navigate to Java perspective and it will show you parseContent function source code. Code piece listed here:

```
protected void parseContent(File file)
{
    try {
        SAXParser parser = createParser();
        InputStream is = new FileInputStream(file);
        if (is == null) {
            return;
        }
        parser.parse(is, this);
    }
    ...
}
```

6. You will find “SAXParser” object will be created by invoking createParser() function each time when we call parseContent(). It can be refactored like this:

```
SAXParser parser = createParser();
protected void parseContent(File file)
{
    try {
        InputStream is = new FileInputStream(file);
        if (is == null) {
            return;
        }
        parser.parse(is, this);
    }
    ...
}
```

7. Run again with profiler. You will find things changed. 😊

## Demo2: MemoryLeak

### Introduction

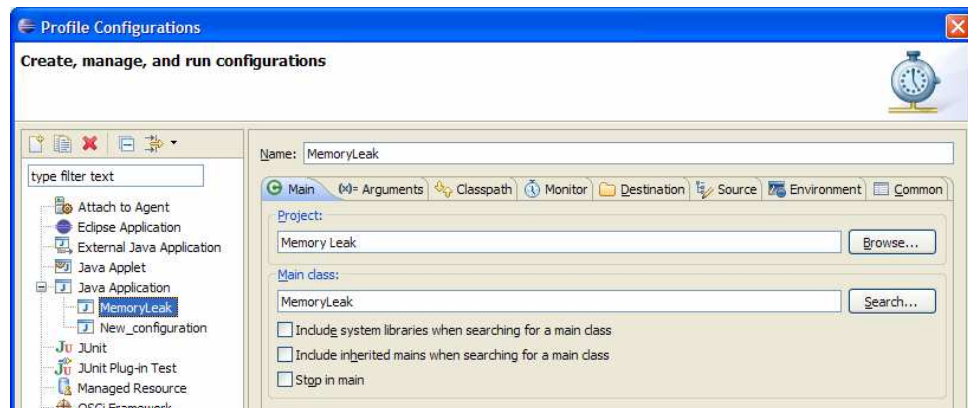
Detect memory leak in “MemoryLeak” application. There’re three buttons will be showed to you: “Leak some memory”, “Allocate some memory”, “Run Garbage Collector”.

### Details:

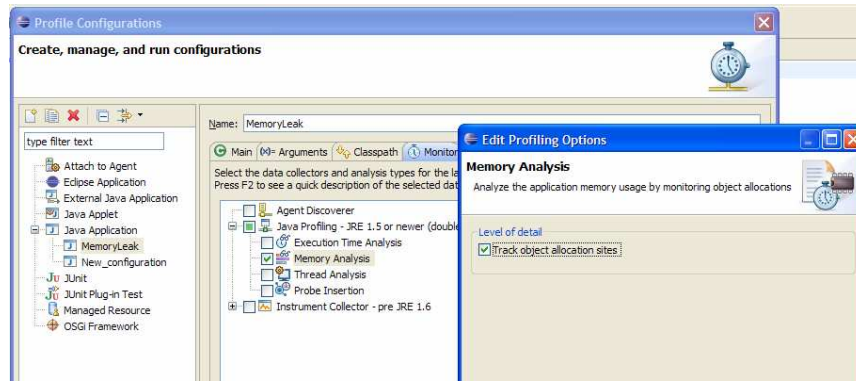
Please open src directory in MemoryLeak project. There’s only 1 file MemoryLeak.java there. MemoryLeak class extend JFrame and will show three buttons. When we press “Leak some memory” button, it will allocate new objects. These objects will be added to list and won’t be used any longer.

### Steps to run:

1. Open “Profile Configurations” Dialog. Double click ”Java Application” in Monitor and Navigator view. Set “Project” to Memory Leak and “Main class” to MemoryLeak



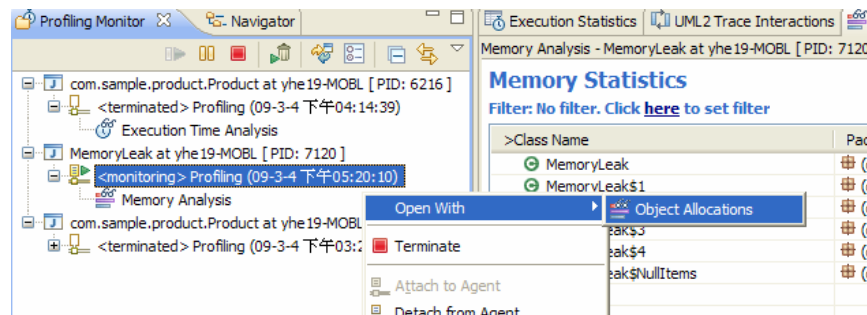
2. Open “Monitor” tab and choose “Memory Analysis” and click “Edit Options” button to make sure you check “track object allocation sites”.



3. Click "Profile" button to profile.
4. Press "Leak some memory" button in application window



5. Open "Profiling and Logging perspective" and view data with "Object Allocation View"

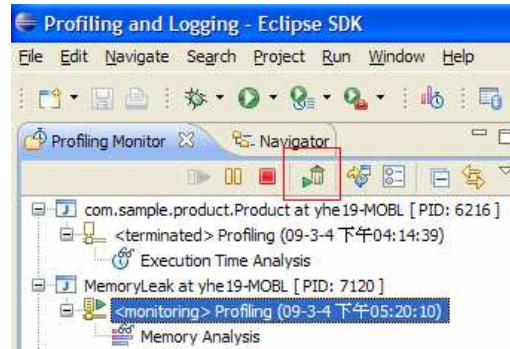


## Steps to identify problem:

1. Press "Leak some memory" button in application window several time to produce garbage.



2. Click on "Run garbage collection" button to force garbage collection events



3. Monitor the age of object, we found that age of several objects continues to grow. Choose these objects as candidates. Since instances number of MemoryLeak\$NullItems are very large. It can be a good candidates.

Class Name	Package	Live Instan...	Active Size...	Total Insta...	Total Size (...)	<Avg. Age
MemoryLeak\$1	(default package)	1	16	1	16	19
MemoryLeak\$3	(default package)	1	16	1	16	19
MemoryLeak\$2	(default package)	1	16	1	16	19
MemoryLeak	(default package)	1	376	1	376	19
MemoryLeak\$NullItems	(default package)	15000	240000	15000	240000	17.15
MemoryLeak\$4	(default package)	0	0	1	8	2

4. Open “Java perspective” and find related source code there in MemoryLeak.java.

```

private JButton getBtnLeak() {
    if (btnLeak == null) {
        btnLeak = new JButton();
        btnLeak.setText("Leak some memory");
        btnLeak.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                for (int i = 0; i < 1000; i++)
                {
                    NullItems item = new NullItems();
                    itemsSet.add(item);
                }
            }
        });
    }
    ...
}

```

5. It can be found that objects of NullItems types are allocated and added into set. Then it won't be used again. Memory is leaked.
6. User can refactor the code to eliminate this defects.