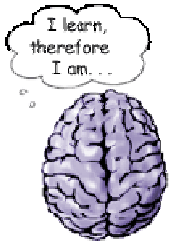
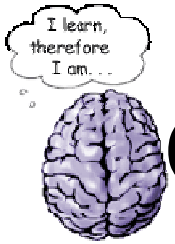
Chapter 5: The Singleton Pattern



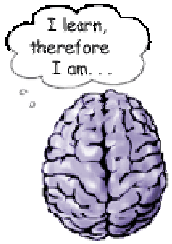
The Chocolate Factory Example

The chocolate factory has computer controlled chocolate boilers. The job of the boiler is to take chocolate and milk, bring them to a boil, and then pass them on to the next phase of making chocolate bars. One of the main functions of the system is to prevent accidents such as draining 500 litres of unboiled mixture, or filing the boiler when is already full, or boiling an empty boiler.



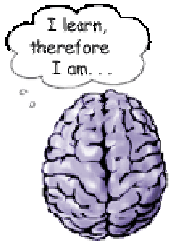
Creating a Single Instance of a Class

- In some cases it maybe necessary to create just one instance of a class.
- This maybe necessary because:
  - More than one instance will result in incorrect program behavior
  - More than one instance will result in the overuse of resources
  - More than one instance will result in inconsistent results
  - There is a need for a global point of access
- How would you ensure that just one instance of a class is created?



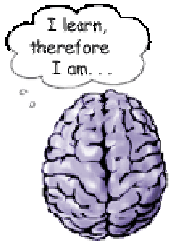
Singleton Pattern Overview

- In some cases there should be at most one instance of a class.
- This one instance must be accessible by all “clients”, e.g. a printer spooler.
- This usually occurs when a global resource has to be shared.
- The singleton pattern ensures that a class has only one instance and provides only one point of entry.



Implementing the Singleton Pattern

- Implement a private constructor to prevent other classes from declaring more than one instance.
- Implement a method to create a single instance. Make this method static.
- Create a lazy instance of the class in the class.
- Make the data element static.



Thread Example

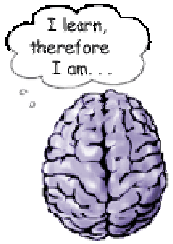
A dual processor machine, with two threads calling the *getInstance()* method for the chocolate boiler

Thread 1

```
public stat ChocolateBoiler
        getInstance()

if (uniqueInstance == null)

uniqueInstance =
       new ChocolateBoiler()

return uniqueInstance;
```

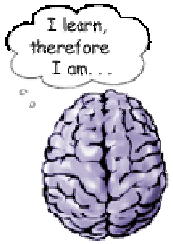
Thread 2

```
public stat ChocolateBoiler
        getInstance()

if (uniqueInstance == null)

uniqueInstance =
       new ChocolateBoiler()

return uniqueInstance;
```



Problems with Multithreading

- In the case of multithreading with more than one processor the getInstance() method could be called at more or less the same time resulting in to more than one instance being created.
- Possible solutions:
  - Synchronize the getInstance() method
  - Do nothing if the getInstance() method is not critical to the application.
  - Move to an eagerly created instance rather than a lazily created one.

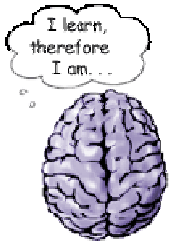


Synchronizing the *getInstance()* Method

- Code

```
public static synchronized Singleton getInstance()
{…
}
```

- Disadvantage – synchronizing can decrease system performance by a factor of 100.

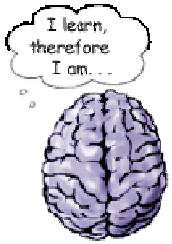


Use an Eagerly Created Instance Rather than a Lazy One

- Code:

```
//Data elements
public static Singleton uniqueInstance = new
   Singleton()

private Singleton() {}

public static Singleton getInstance() {
   return uniqueInstance
}
```

- Disadvantage – Memory may be allocated and not used.



In Summary

- The singleton pattern ensures that there is just one instance of a class.
- The singleton pattern provides a global access point.
- The pattern is implemented by using a private constructor and a static method combined with a static variable.
- Possible problems with multithreading.
- Versions of Java earlier that 1.2 automatically clear singletons that are not being accessed as part of garbage collection.