Design

Introduction

- The design of a software system is the specification of the architecture that will be implemented in its code in order to fulfil the functional and the performance requirements.
- It is highly important to complete the design before we start coding.
- Each programming language has its own characteristics. The design should utilize the characteristics of the programming language we intend to use.

Abstraction

- Coding our program while using classes without knowing the exact implementation will simplify our work.
- Define abstract classes will allow us an higher level of abstraction.

Reuse

- Taking into consideration existing code and reusing it when appropriate.
- Writing reusable code will enable us to code shorter programs and simpler to maintain.

First Steps

- Doing our first steps designing a software system it would be useful to start with dividing our software system into sub systems.
- For each sub system it would be useful to follow with choosing its threading model, specifying its classes, structures and algorithms.

Object Orietned Perspective

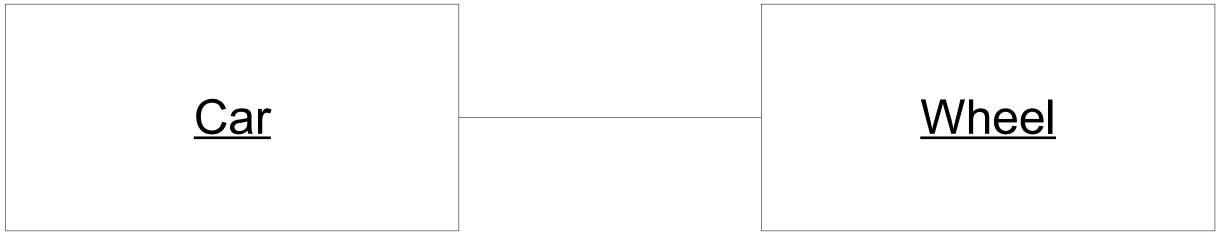
- In case that you are not familiar with object oriented programming it is highly important making the effort and conceive the system with an object oriented perspective and ensure that you don't mistakenly think procedural.
- If you are not familiar with object oriented programming try to think in terms of classes, properties, behaviors and components.

Overobjectification

- There is often a fine line between a proper and a creative object oriented system and an annoying one. Turning each and every little thing into an object might be annoying.

The Has-A Relationship

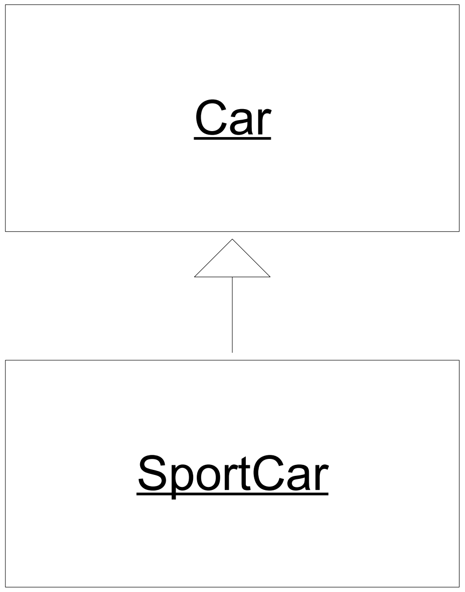
- When objects engaged in a has-a (aggregation) relationship we can envisionone object as part of another.



The Car Has a Wheel

The Is-A Relationship

- When objects engaged in a is-a (inheritance) relationship we can envision one object as if it is like the other. We can conceive the class from which the first object was instantiated as one that extends the other one.



The SportCar is a Car

Multiple Inheritance

- Unlike Java and C#, the C++ programming language allows us to maintain multiple inheritances.
- We better avoid multiple inheritance as it might cause complexities and might damage the clarity of our program.

Abstract Class

- The C++ programming language doesn't allow us to define interfaces as Java and C#. Instead, we should define abstract classes from which the concrete ones will inherit.
- This way we can achieve a clear separation between the interface and the implementation.

Reusing Code

- When reusing code there are more than a few issues to take into consideration. Is the code safe for multithreading? Are there any specific initializations or cleanups? Are there other libraries the code depends on?

The Big-O Notation

- The Big-O notation specifies a relative performance assesement. It doesn't provide us with an absolute measurement.
- Knowing the Big-O assesment for the third party library we choose we can pick the right one for our program needs.

Open Source Libraries

- The use of open source libraries has become tremendeously popular during the last years.
- When using open source libraries make an effort to stick with the "freedom" philosophy.

The C++ Standard Library

- Mostlikely this is the most important library in use when coding in C++.
- This library is part of the C++ standard. We can find it within every standard compiler.
- The STL provides an implementation for a range of generic algorithms, such as the search and the sorting algorithms.

Design Techniques

- A design technique is a standard approach for solving particular problems in C++.

  The smart pointers is one of the most common design techniques. Smart pointer is an object that contains a pointer. The smart pointer is allocated on the stack so that once it goes out of scope its destructor takes care of deleting the contained pointer.

Design Patterns

- A design pattern is a standard approach for solving a general problem.
- Comparing with design techniues, the design patterns are less language specific.

Software Life Cycle Models

- The need for a formalized process yields several different approaches.
- The most popular of them include the following: Stagewise Model, Waterfall Model and the Spiral Method.

Software Engineering Methodologies

- The software engineering methodologies provide practical rules of thumb for professional software development.

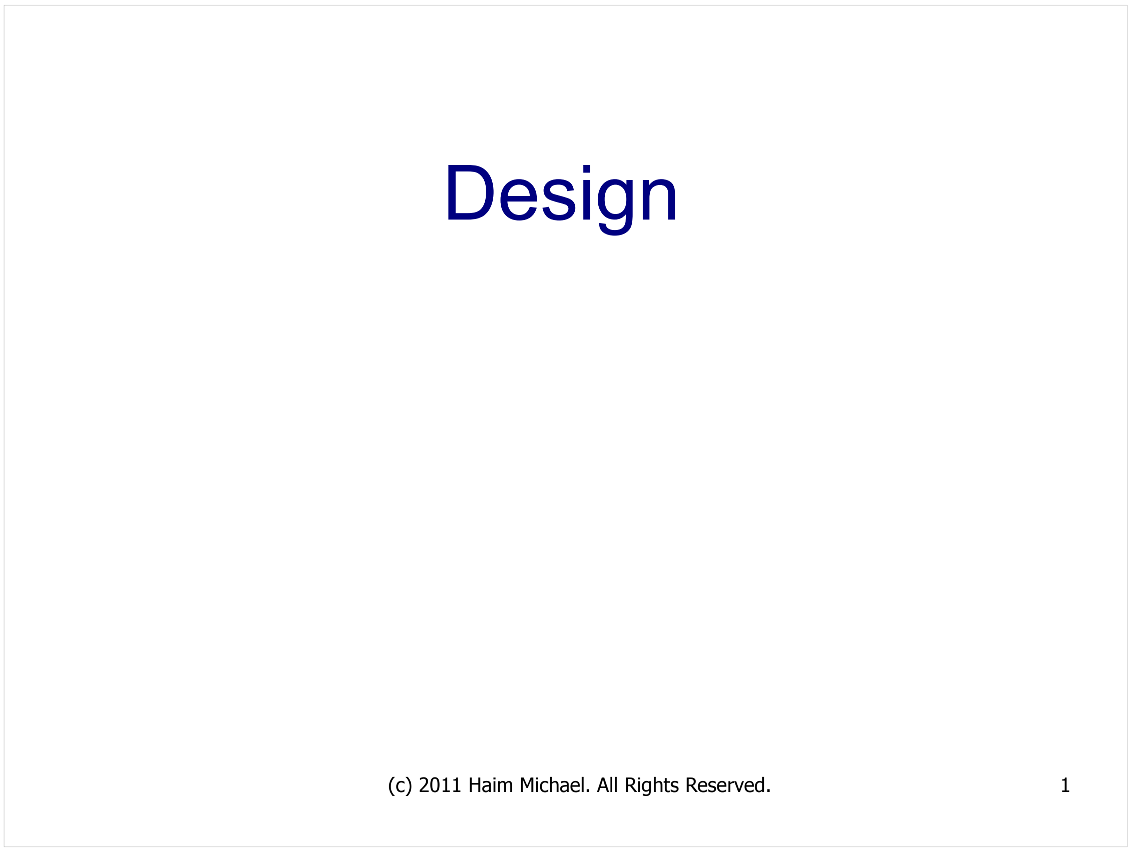
Coding with Style

- It isn't just about coding. It is important that our work will look good.
- Keep the code clear and understandable. Make sure your code includes proper comments and documentation. Select good names in according with the conventions in your work.
- Use constants when possible. It will ease the maintenance of your code in the long run.

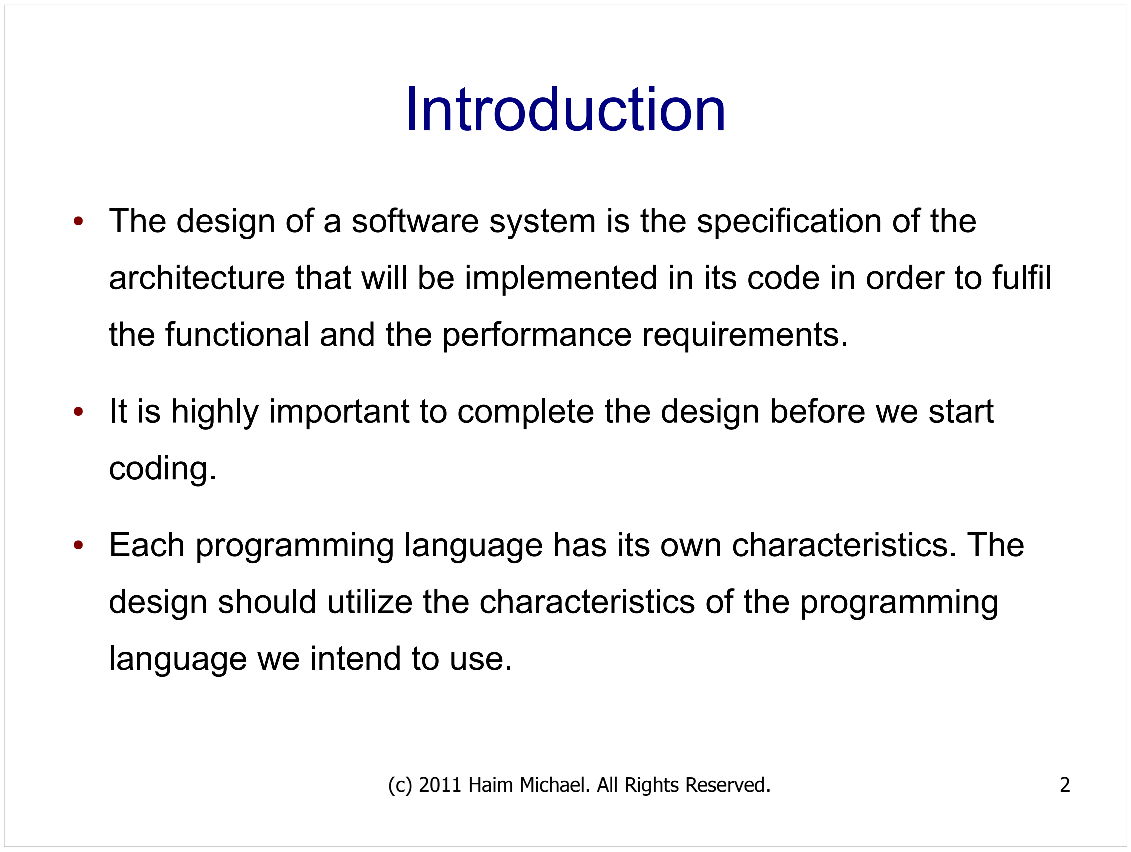
Coding with Style

- Whenever a statement is required (e.g. condition statements) prefer using a compound statement, even when a simple one is sufficient.

```
...
if(num>100)
{
    std::cout << "do this... do that...";
}
...
```

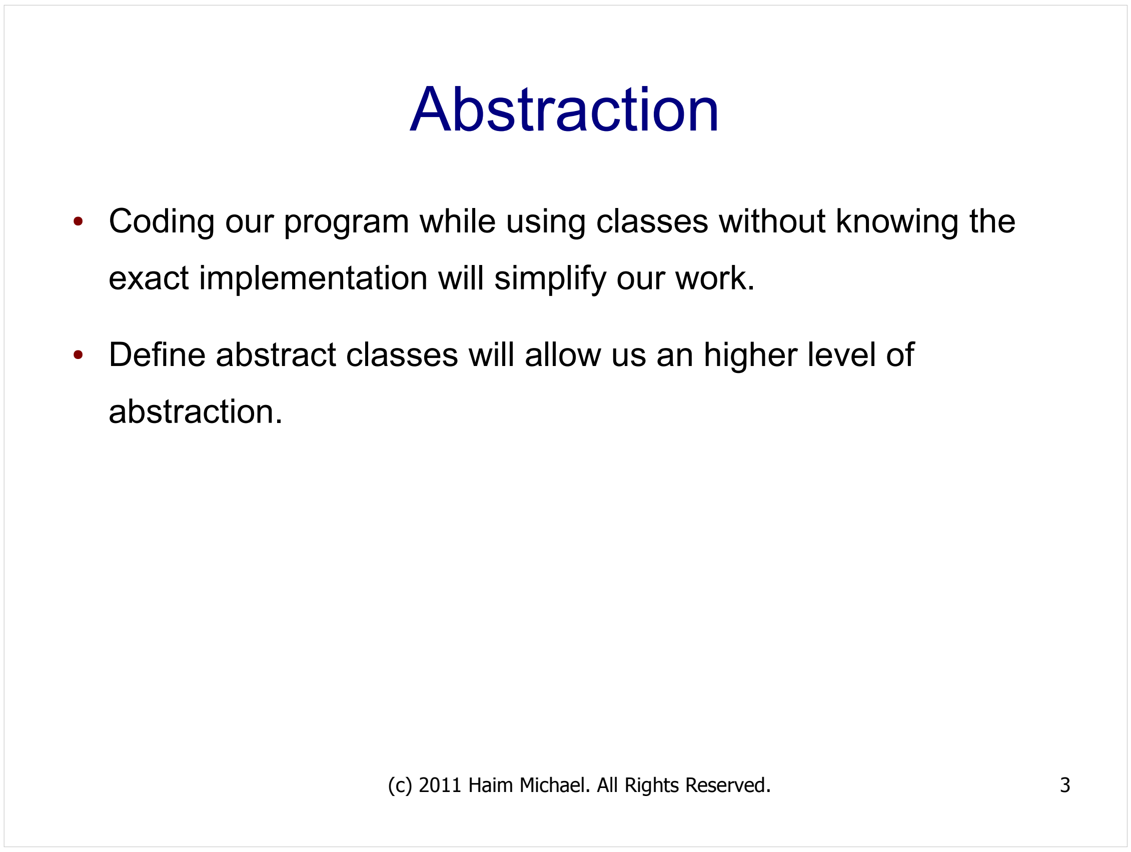


Design



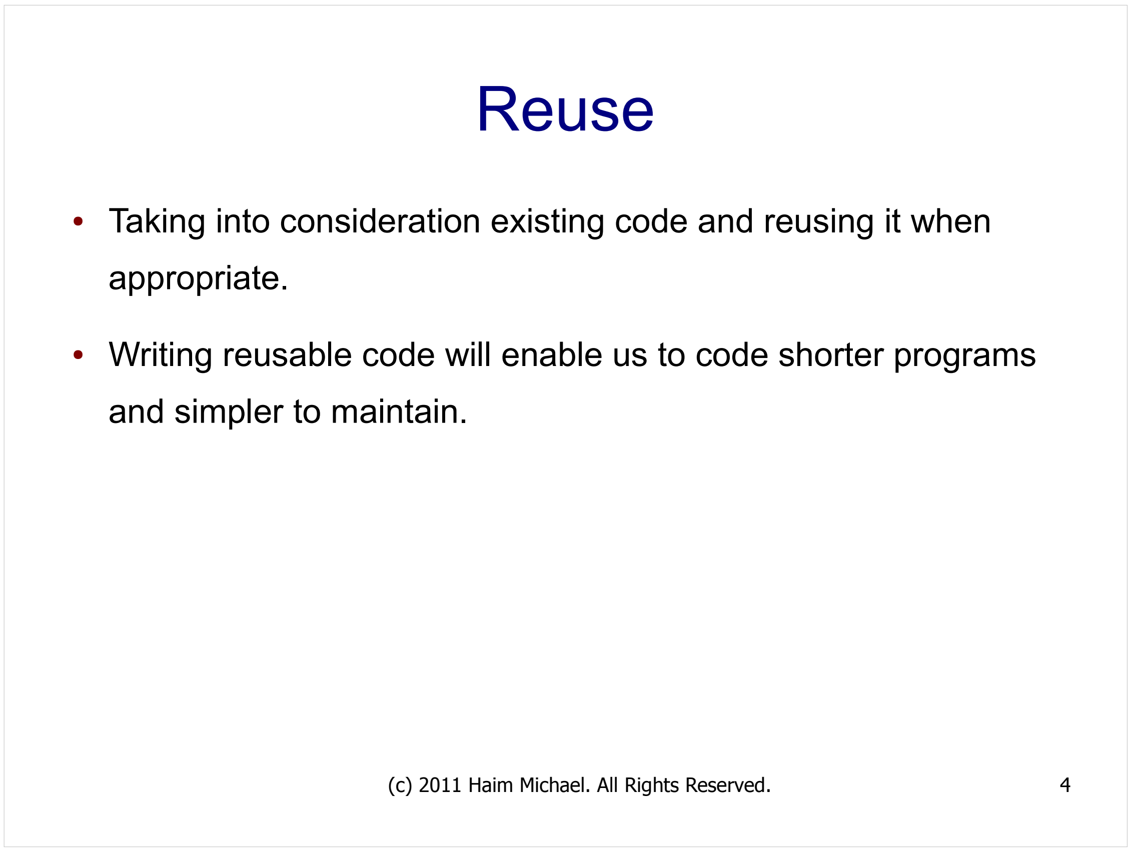
Introduction

- The design of a software system is the specification of the architecture that will be implemented in its code in order to fulfil the functional and the performance requirements.
- It is highly important to complete the design before we start coding.
- Each programming language has its own characteristics. The design should utilize the characteristics of the programming language we intend to use.



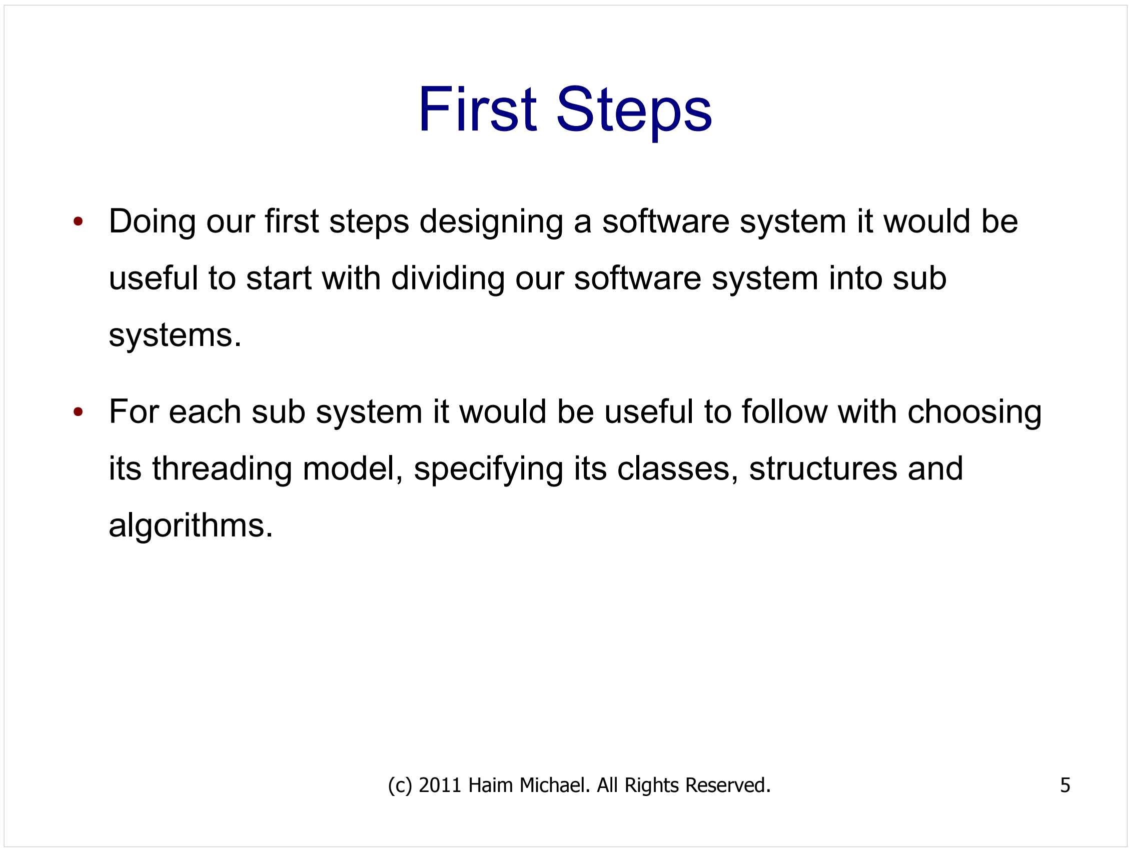
Abstraction

- Coding our program while using classes without knowing the exact implementation will simplify our work.
- Define abstract classes will allow us an higher level of abstraction.



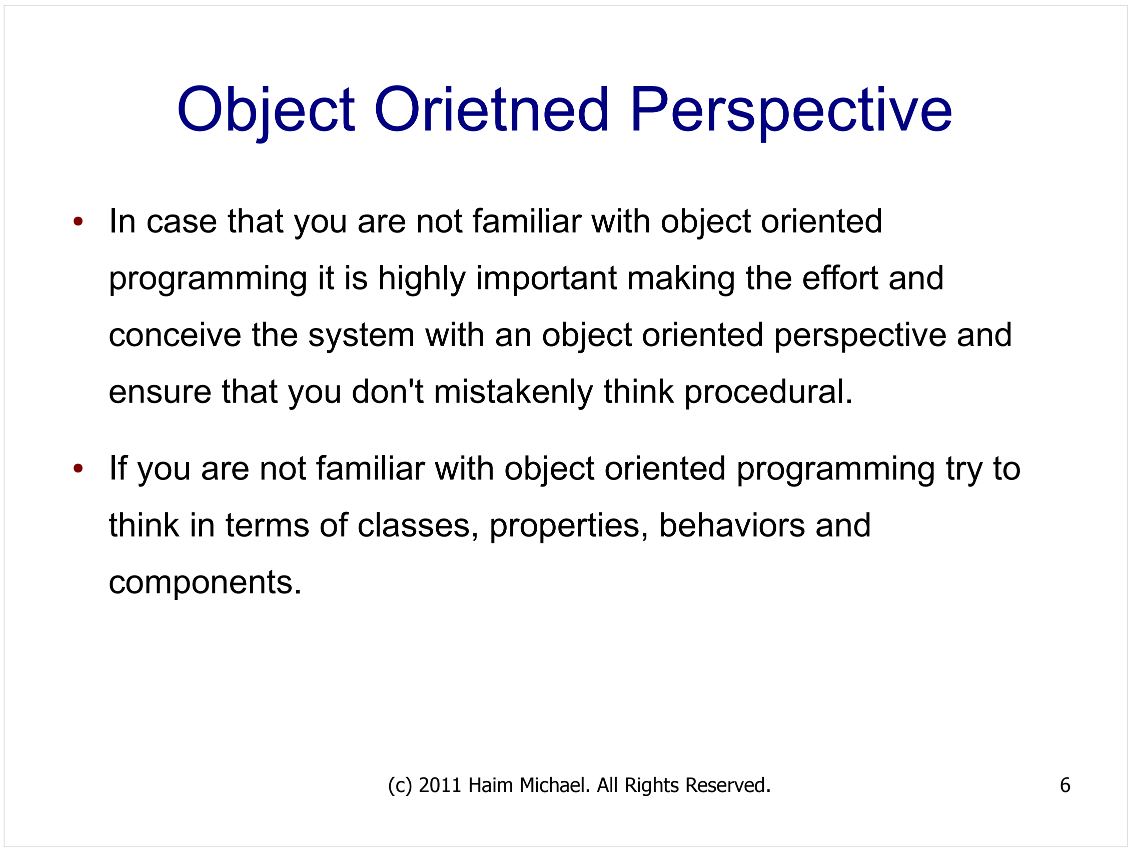
Reuse

- Taking into consideration existing code and reusing it when appropriate.
- Writing reusable code will enable us to code shorter programs and simpler to maintain.



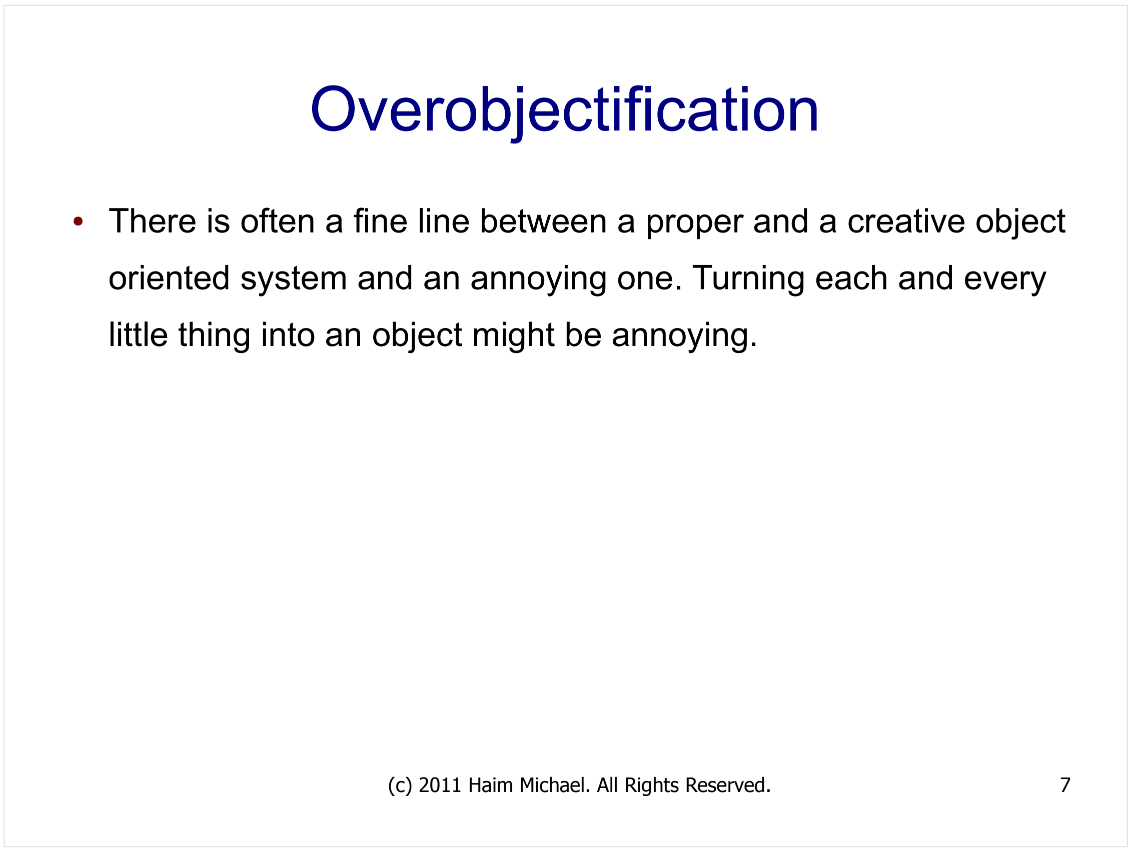
First Steps

- Doing our first steps designing a software system it would be useful to start with dividing our software system into sub systems.
- For each sub system it would be useful to follow with choosing its threading model, specifying its classes, structures and algorithms.



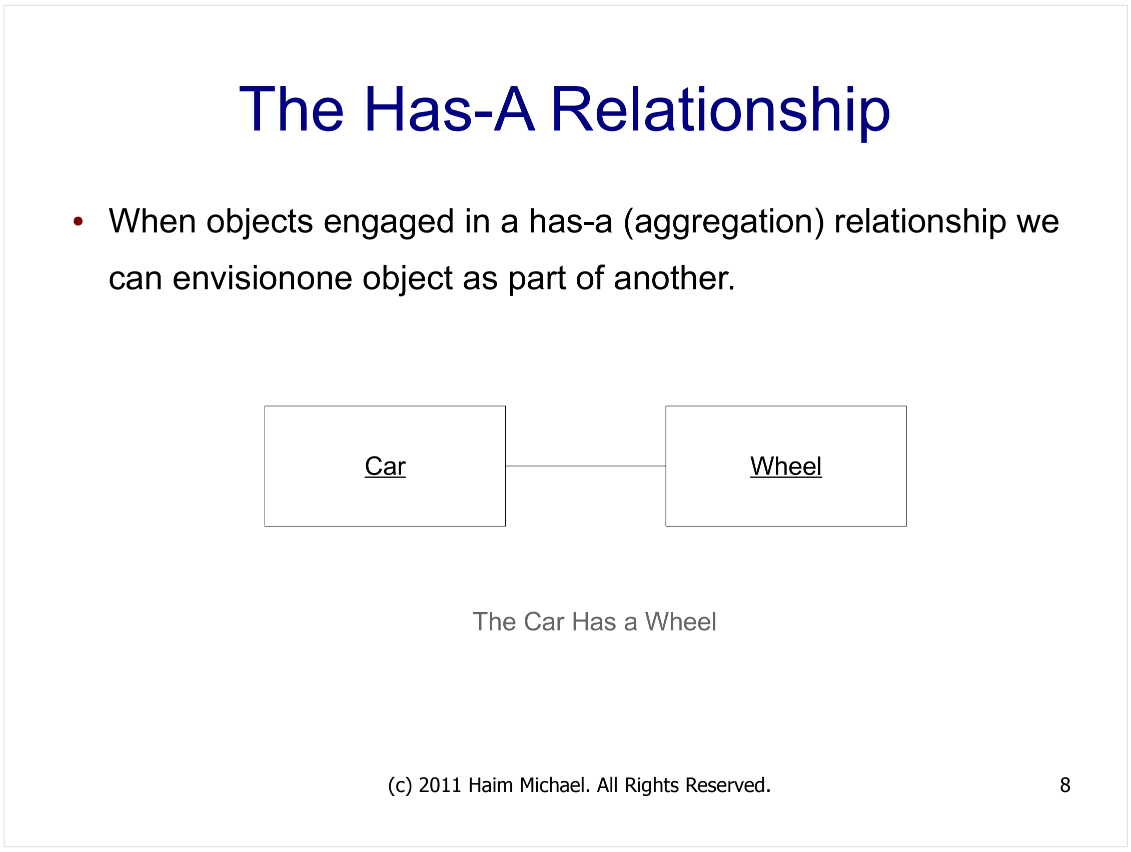
Object Orietned Perspective

- In case that you are not familiar with object oriented programming it is highly important making the effort and conceive the system with an object oriented perspective and ensure that you don't mistakenly think procedural.
- If you are not familiar with object oriented programming try to think in terms of classes, properties, behaviors and components.



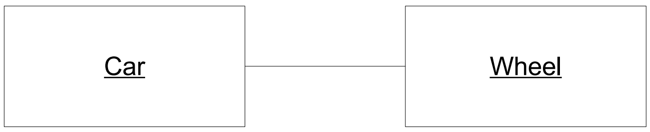
Overobjectification

- There is often a fine line between a proper and a creative object oriented system and an annoying one. Turning each and every little thing into an object might be annoying.

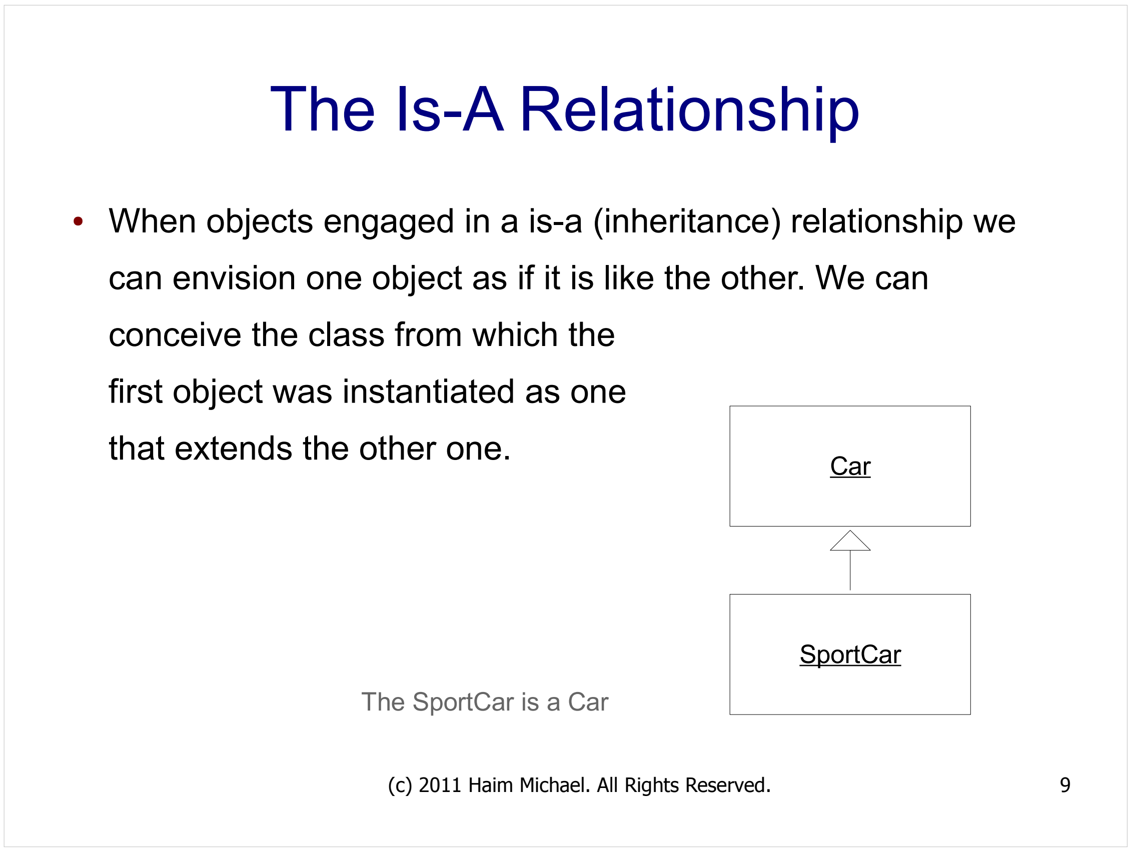


The Has-A Relationship

- When objects engaged in a has-a (aggregation) relationship we can envisionone object as part of another.

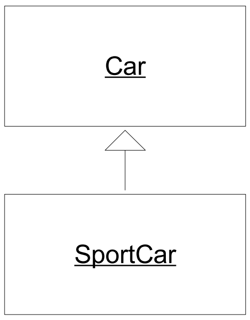


The Car Has a Wheel

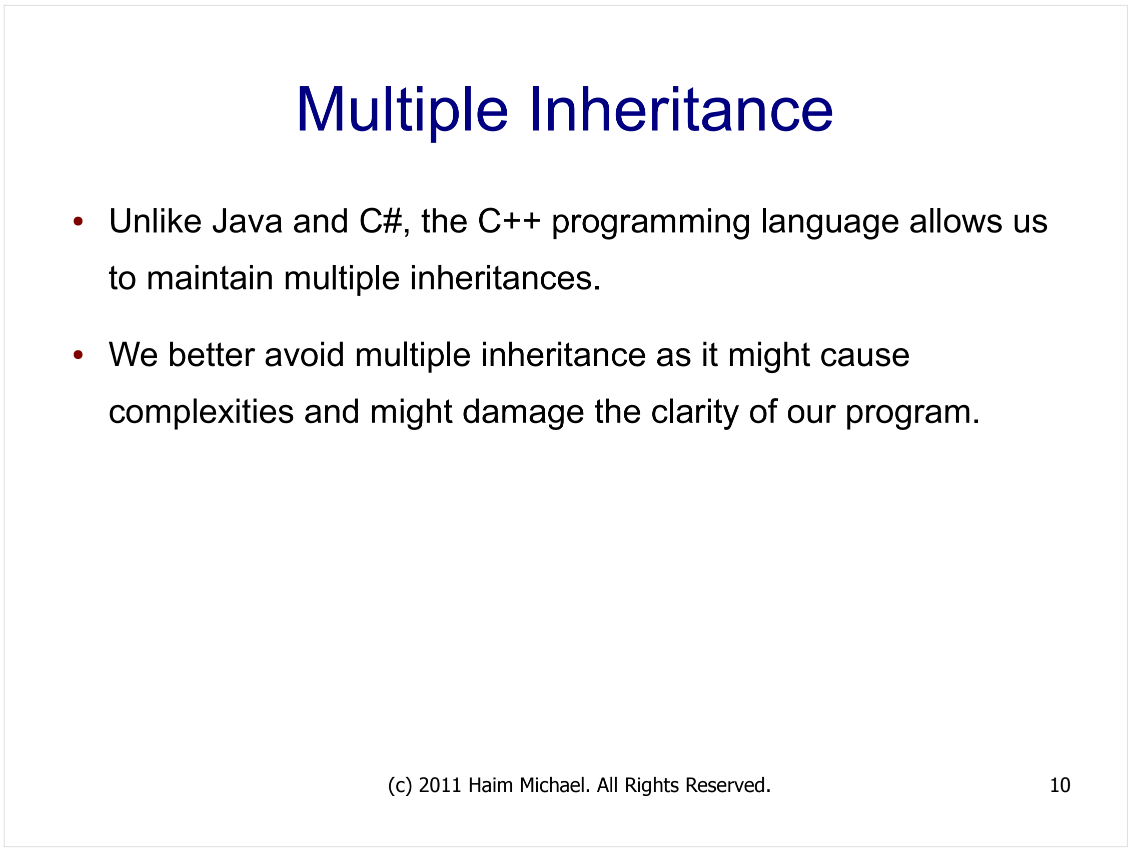


The Is-A Relationship

- When objects engaged in a is-a (inheritance) relationship we can envision one object as if it is like the other. We can conceive the class from which the first object was instantiated as one that extends the other one.



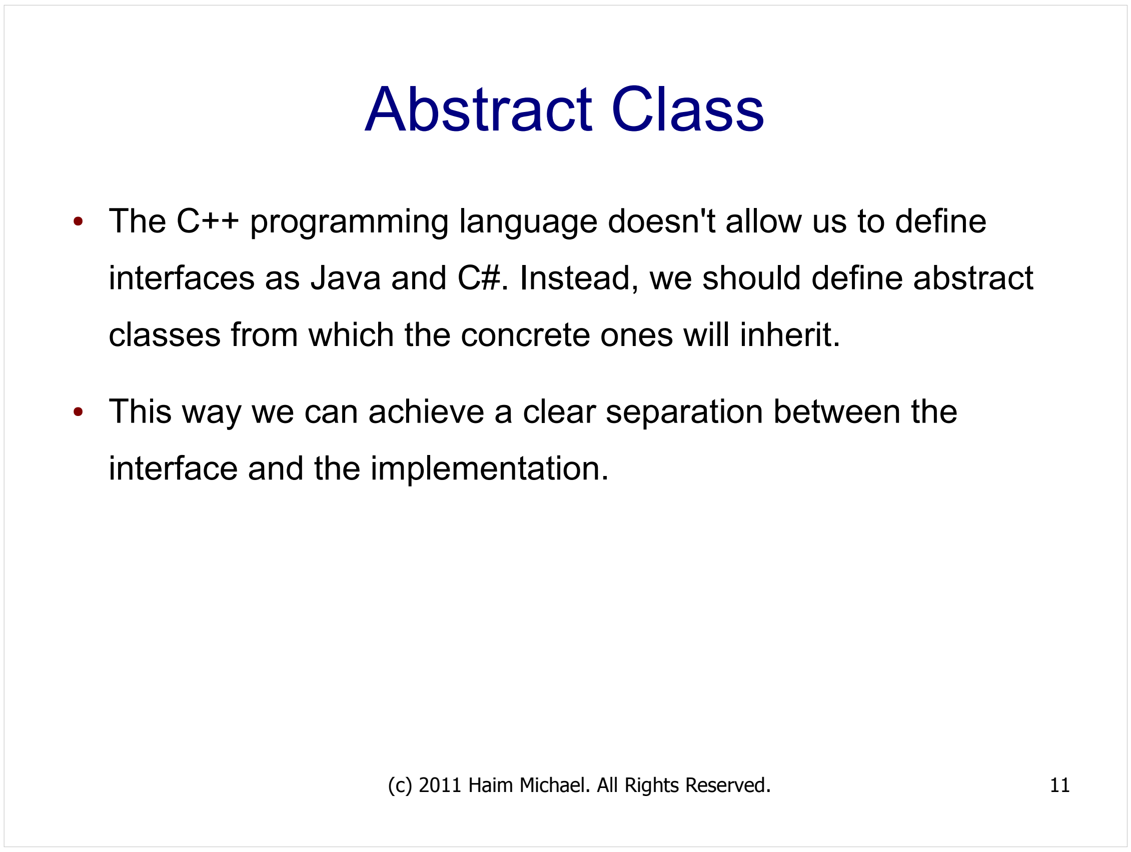
The SportCar is a Car



Multiple Inheritance

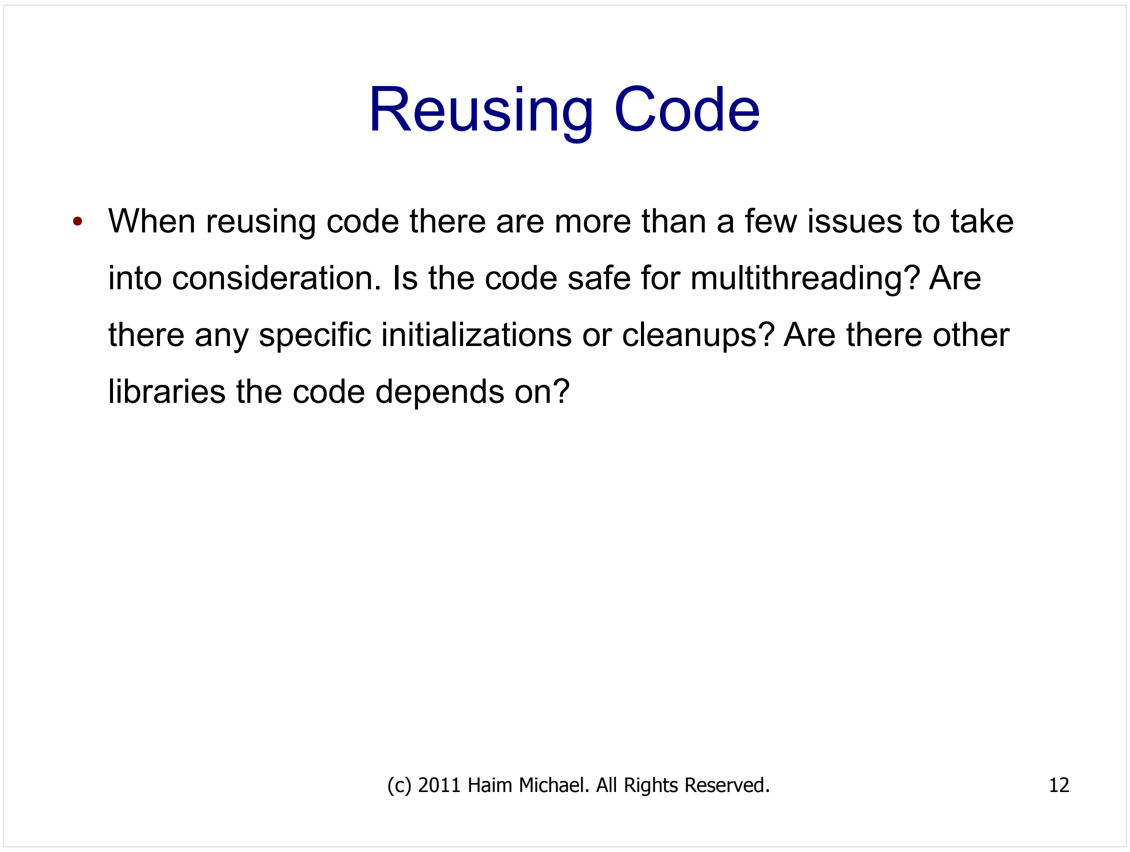
- Unlike Java and C#, the C++ programming language allows us to maintain multiple inheritances.
- We better avoid multiple inheritance as it might cause complexities and might damage the clarity of our program.

10



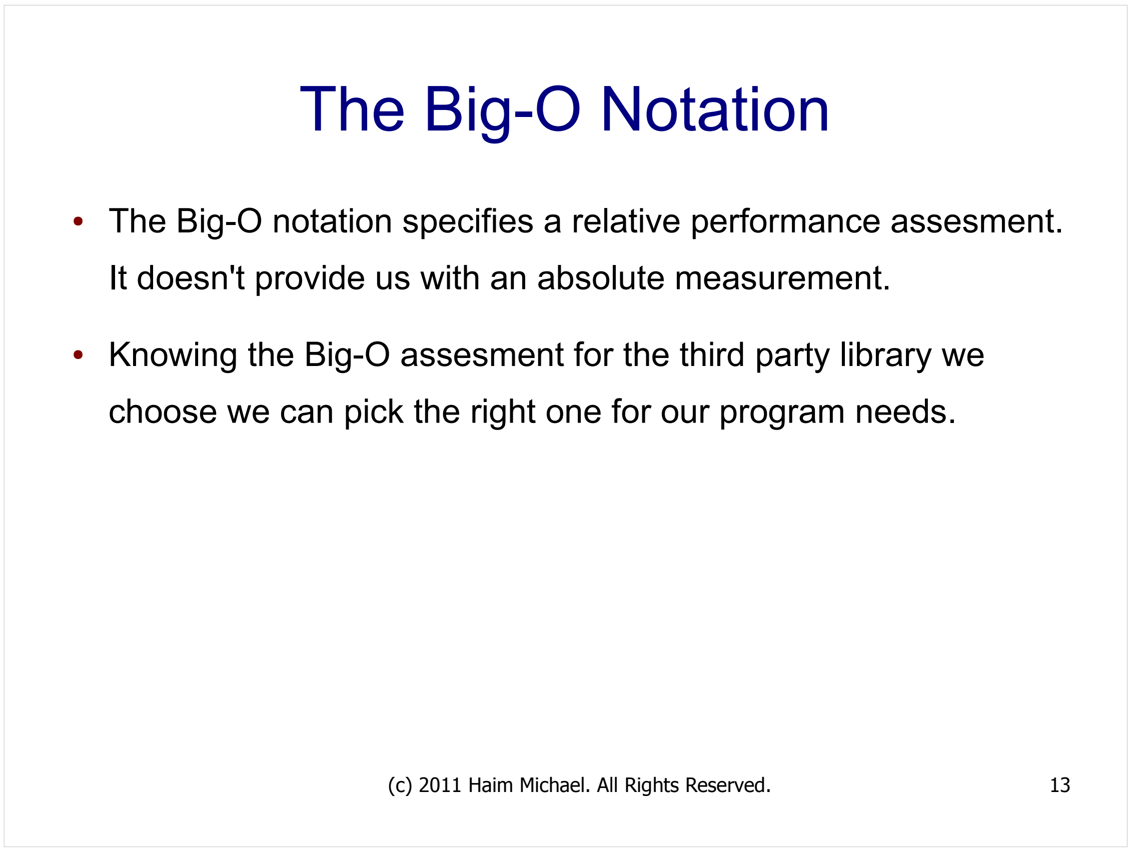
Abstract Class

- The C++ programming language doesn't allow us to define interfaces as Java and C#. Instead, we should define abstract classes from which the concrete ones will inherit.
- This way we can achieve a clear separation between the interface and the implementation.



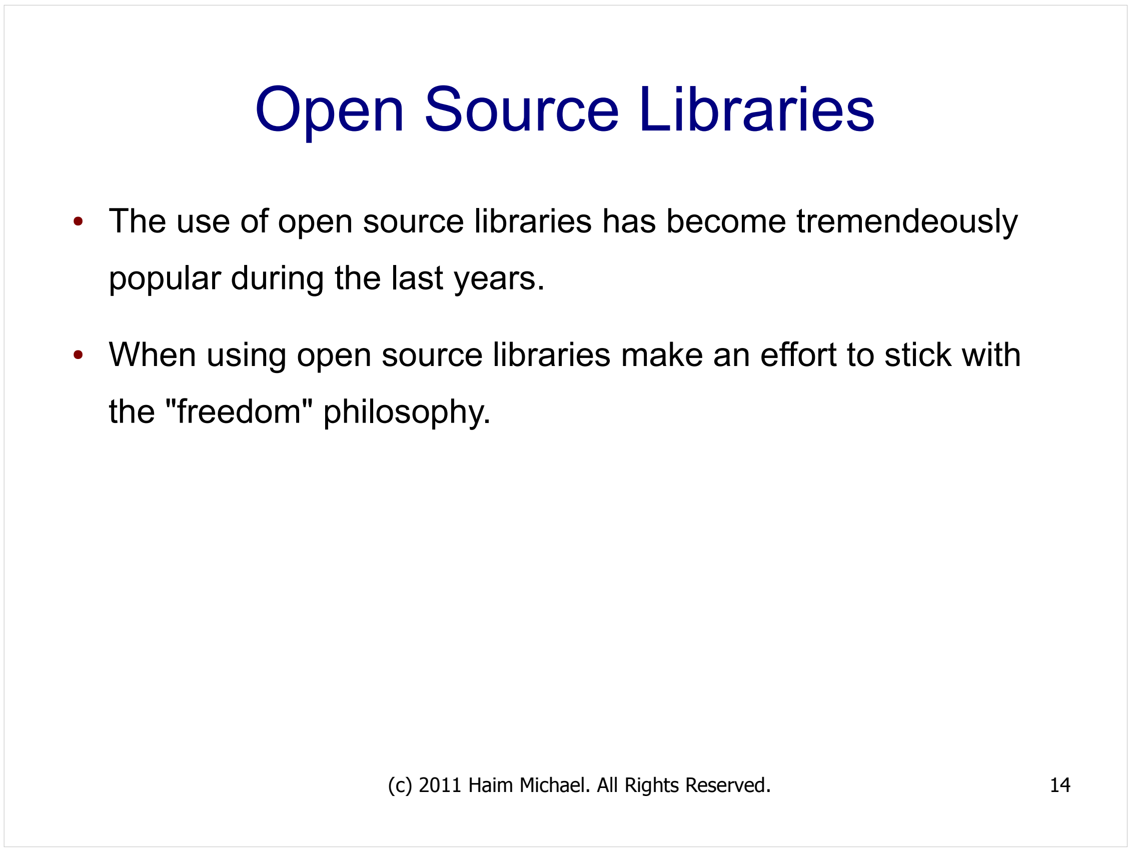
Reusing Code

- When reusing code there are more than a few issues to take into consideration. Is the code safe for multithreading? Are there any specific initializations or cleanups? Are there other libraries the code depends on?



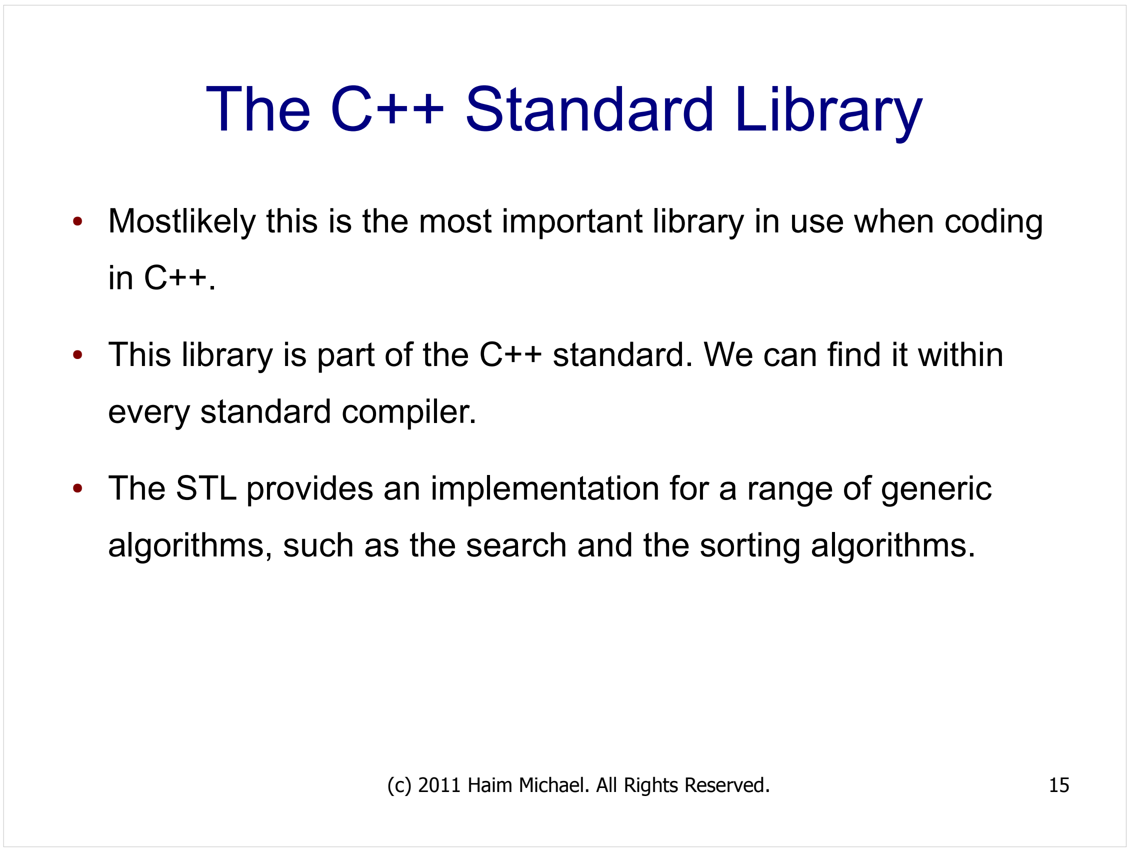
The Big-O Notation

- The Big-O notation specifies a relative performance assesment. It doesn't provide us with an absolute measurement.
- Knowing the Big-O assesment for the third party library we choose we can pick the right one for our program needs.



Open Source Libraries

- The use of open source libraries has become tremendeously popular during the last years.
- When using open source libraries make an effort to stick with the "freedom" philosophy.



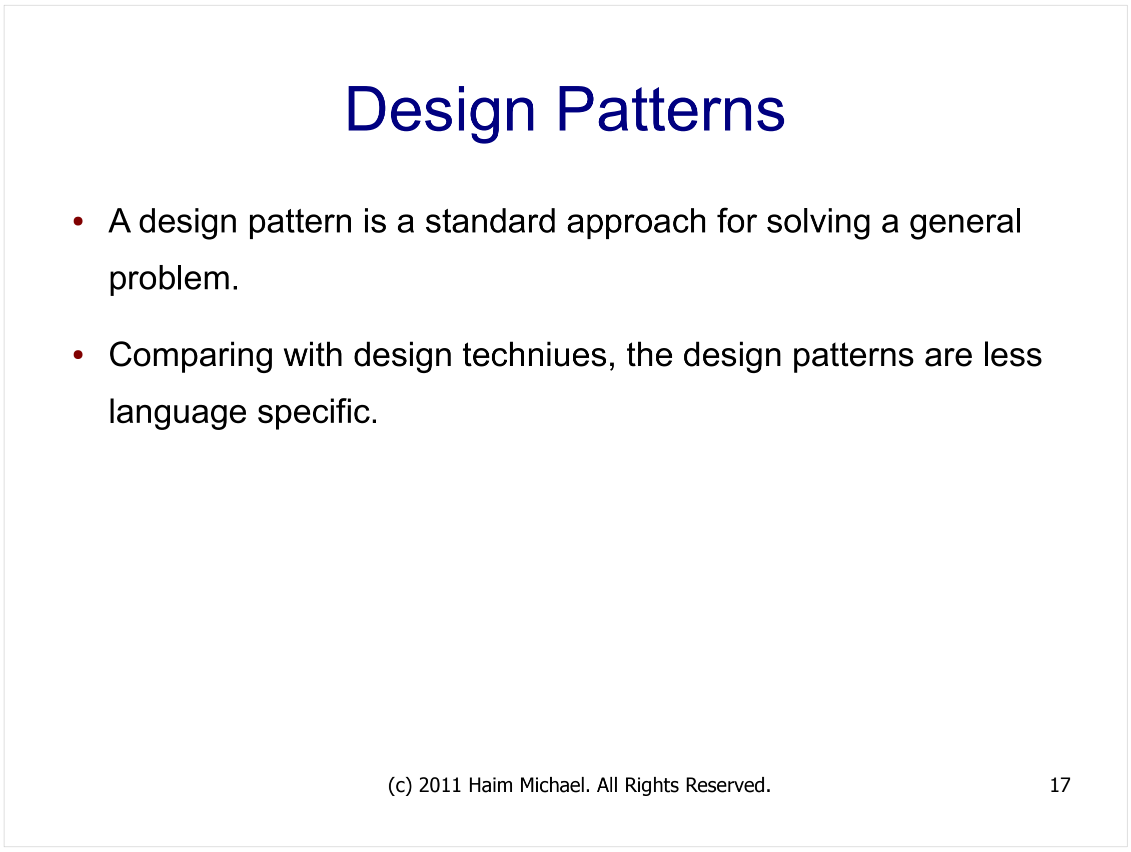
The C++ Standard Library

- Mostlikely this is the most important library in use when coding in C++.
- This library is part of the C++ standard. We can find it within every standard compiler.
- The STL provides an implementation for a range of generic algorithms, such as the search and the sorting algorithms.

Design Techniques

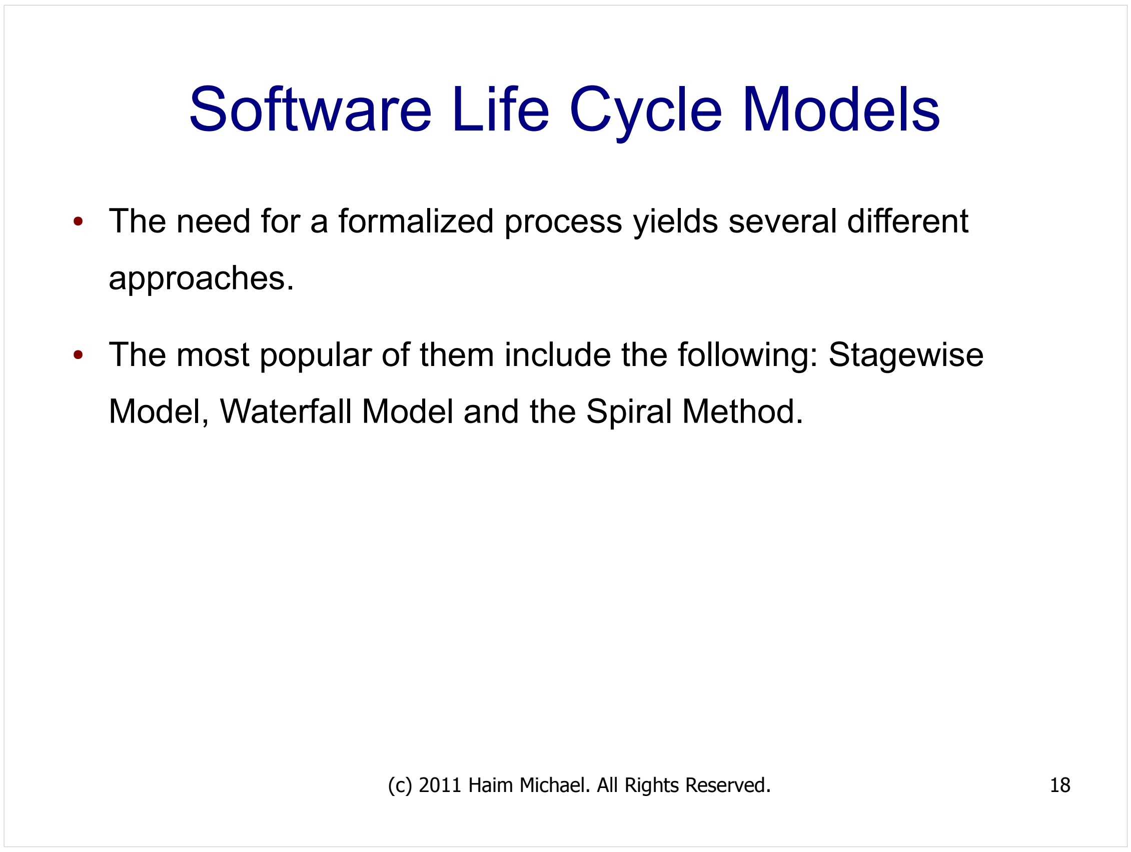
- A design technique is a standard approach for solving particular problems in C++.

  The smart pointers is one of the most common design techniques. Smart pointer is an object that contains a pointer. The smart pointer is allocated on the stack so that once it goes out of scope its destructor takes care of deleting the contained pointer.



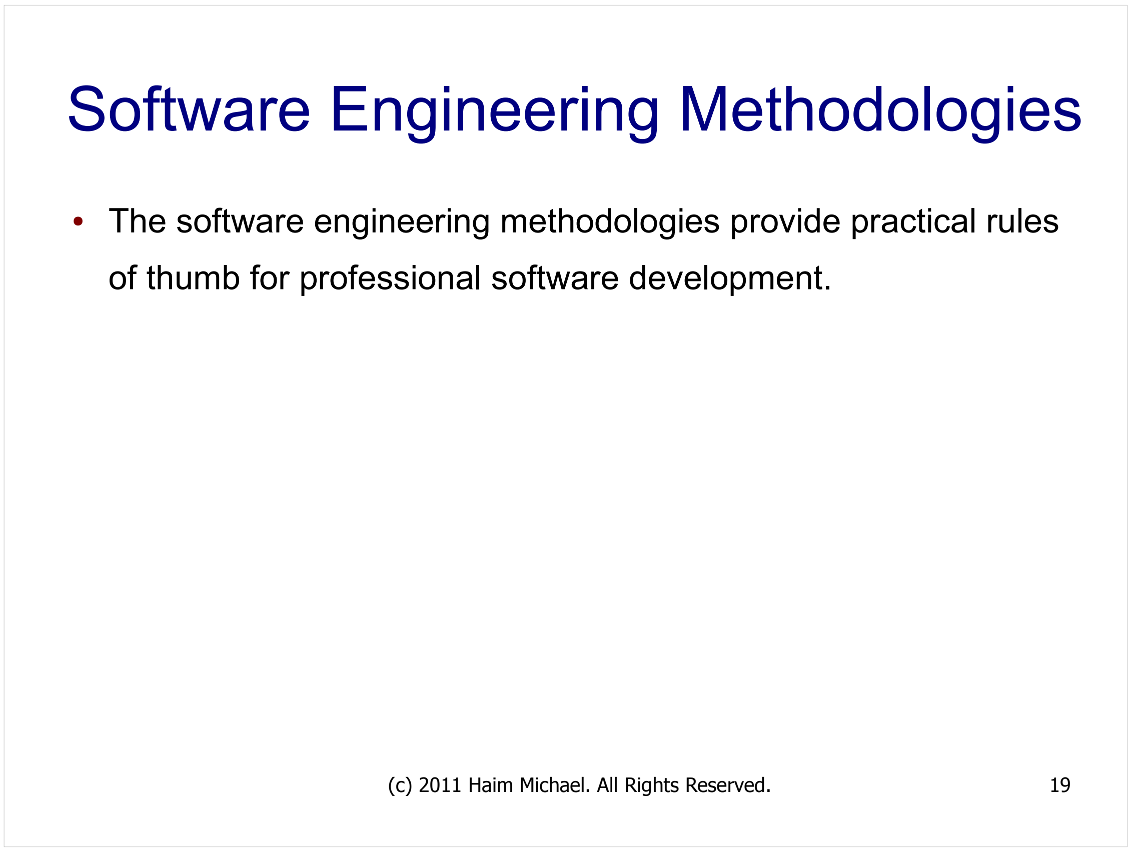
Design Patterns

- A design pattern is a standard approach for solving a general problem.
- Comparing with design techniues, the design patterns are less language specific.



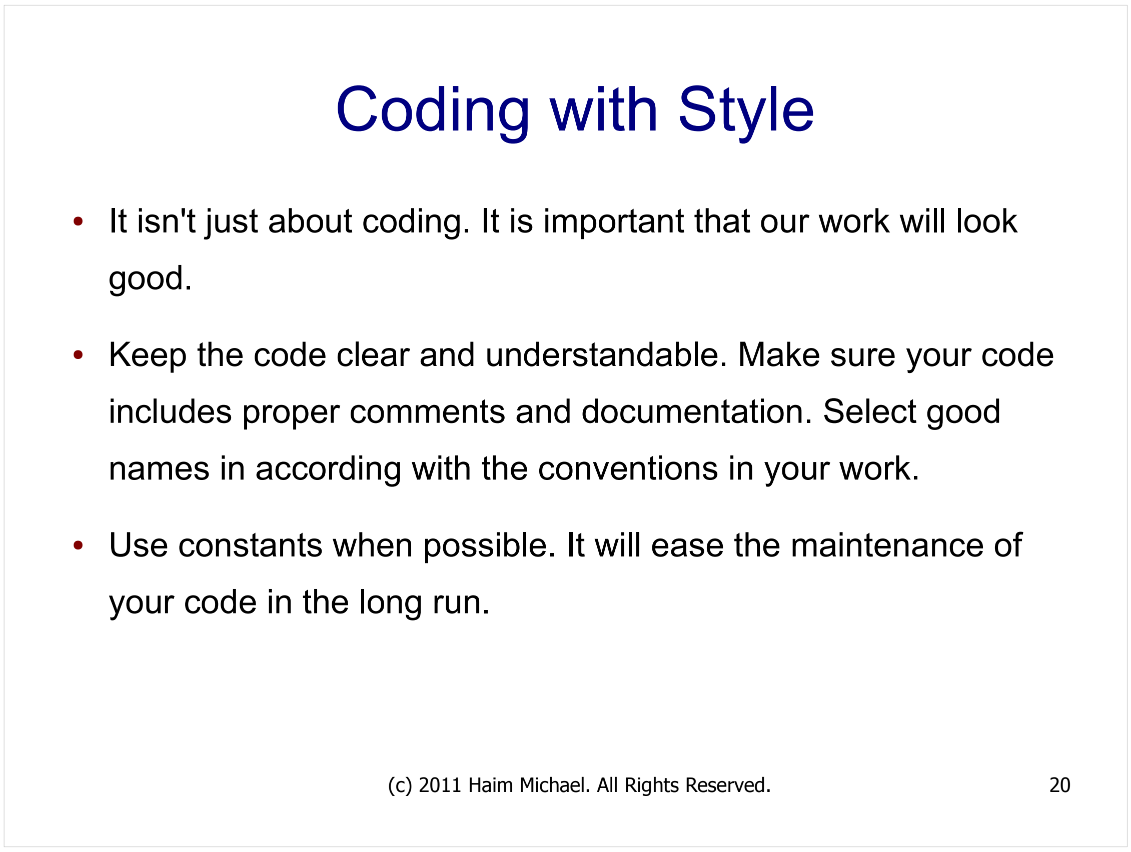
Software Life Cycle Models

- The need for a formalized process yields several different approaches.
- The most popular of them include the following: Stagewise Model, Waterfall Model and the Spiral Method.



Software Engineering Methodologies

- The software engineering methodologies provide practical rules of thumb for professional software development.



Coding with Style

- It isn't just about coding. It is important that our work will look good.
- Keep the code clear and understandable. Make sure your code includes proper comments and documentation. Select good names in according with the conventions in your work.
- Use constants when possible. It will ease the maintenance of your code in the long run.

Coding with Style

- Whenever a statement is required (e.g. condition statements) prefer using a compound statement, even when a simple one is sufficient.

```
...
if(num>100)
{
    std::cout << "do this... do that...";
}
...
```