



Informatics II for engineering sciences (MSE)

**From C to Java: An Introduction into
Object Oriented Programming**



Goal

- A gentle Introduction in OO Programming
- Assumption: You know C
 - C is an imperative Language
 - you can even write object oriented code in C
 - We use Java

First Challenge – From C to Java



Session Outline

Session #1

- Getting Started (1)
 - Hello World
 - Primitive Data types
- Getting Started (2)
 - Simple Input / Output
 - Control Structures
- Getting Started (3)
 - Classes in Java, Initialization, Class vs. Instance
 - Information Hiding (public, private, protected)

Session #2

- Getting Started (4)
 - Built-In Classes
 - Type Casting vs. Generics
- Getting Started (5)
 - Java's Object Hierarchy
 - Inheritance vs. Polymorphism
 - Abstract Classes / Interfaces
- Getting Started (6)
 - Parameter Passing
 - Exception Handling



Session Outline

Session #1

- Getting Started (1)
 - Hello World
 - Primitive Data types
- Getting Started (2)
 - Simple Input / Output
 - Control Structures
- Getting Started (3)
 - Classes in Java, Initialization, Class vs. Instance
 - Information Hiding (public, private, protected)

Session #2

- Getting Started (4)
 - Built-In Classes
 - Type Casting vs. Generics
- Getting Started (5)
 - Java's Object Hierarchy
 - Inheritance vs. Polymorphism
 - Abstract Classes / Interfaces
- Getting Started (6)
 - Parameter Passing
 - Exception Handling



Session #1 – Detailed Outline

Getting Started (1)

- Writing your main-Method - Hello World
- Primitive data types in Java
- Exercise #1:
 - Setup your Development Environment
 - Create your first Project
 - Create and run your main() method



From C to Java: Getting Started (1)

Main method

In C you learned how write your main method and compile it using gcc.

```
#include <stdio.h>
int main(int argc, char **argv, char **envp)
{
    printf( "\nHello World\n\n" );
    return 0;
};
```

```
gcc main.c -o main
```



From C to Java: Getting Started (1)

Hello World

In Java your main method looks pretty similar

```
public class Main {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

public and static modifiers are explained later during the course

Hint: Can't recall existing specs refer to: <http://docs.oracle.com/javase/7/docs/api/>



From C to Java: Getting Started (1)

Primitive Data Types

Java supports many different data types (1):

// 8 bit signed integer -128 to 127 (inclusive)
byte b = 127;

//16-bit signed integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive)
short shortnumber = 10;

//32-bit signed two's integer, which has a minimum value of -2pow(31) and a maximum value of 2pow(31)-1
int i = 10;

//is a 64-bit two's complement integer. The signed long has a minimum value of -2pow(63) and a max. 2pow(63)-1
long longNumber = 10;

//is a single-precision 32-bit IEEE 754 floating point
float singlePrecision = 2000;

//is a double-precision 64-bit IEEE 754 floating point
double doublePrecision = 20.0;

//The boolean data type has only two possible values: true and false
boolean bool = true;

//is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a
// maximum value of '\uffff' (or 65,535 inclusive).
char character = 'A';

(1) <http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>



From C to Java: Getting Started (1)

Exercise #1

- Setup your Development Environment
- Create your first Project
- Create and run your main() method



From C to Java: Getting Started (1)

Exercise #1

- **Setup your Development Environment (5min)**
- Create your first Project (2min)
- Create and run your main() method (2min)

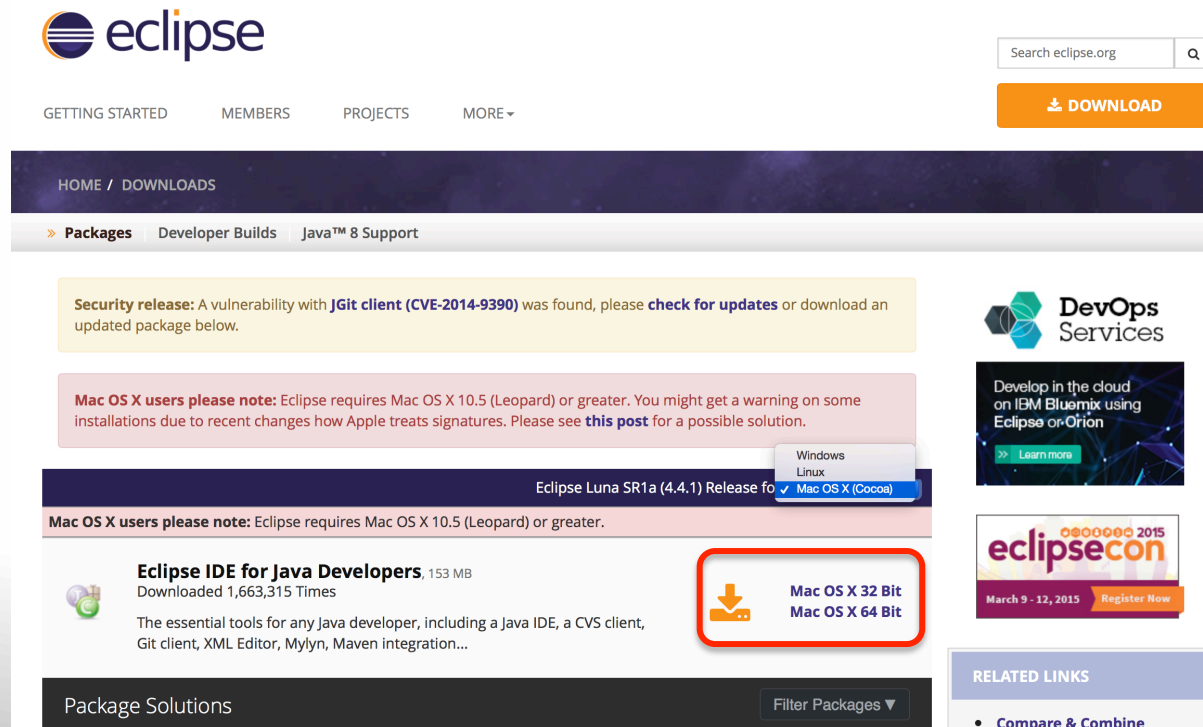


From C to Java: Getting Started (1)

Exercise #1

We use / recommend Eclipse

<http://www.eclipse.org/downloads/>



The screenshot shows the Eclipse website's download page. At the top is the Eclipse logo and a search bar. Below the logo are navigation links: GETTING STARTED, MEMBERS, PROJECTS, and MORE. A prominent orange 'DOWNLOAD' button is on the right. A dark banner below the navigation links reads 'HOME / DOWNLOADS'. Underneath, there are links for 'Packages', 'Developer Builds', and 'Java™ 8 Support'. A yellow box contains a 'Security release' notice about a vulnerability in JGit. Below that, a pink box contains a 'Mac OS X users please note' warning. The main content area features a dropdown menu for operating systems (Windows, Linux, Mac OS X (Cocoa)) with 'Mac OS X (Cocoa)' selected. Below the dropdown, another 'Mac OS X users please note' warning is displayed. The primary download option is 'Eclipse IDE for Java Developers', which is 153 MB and has been downloaded 1,663,315 times. It is described as 'The essential tools for any Java developer, including a Java IDE, a CVS client, Git client, XML Editor, Mylyn, Maven integration...'. To the right of this package, a red box highlights the download links for 'Mac OS X 32 Bit' and 'Mac OS X 64 Bit'. On the right side of the page, there are three promotional banners: 'DevOps Services', 'Develop in the cloud on IBM Bluemix using Eclipse or Orion', and 'eclipsecon 2015'. At the bottom, there is a 'RELATED LINKS' section with a link to 'Compare & Combine' and a 'Package Solutions' section with a 'Filter Packages' dropdown.

eclipse

Search eclipse.org

GETTING STARTED MEMBERS PROJECTS MORE

DOWNLOAD

HOME / DOWNLOADS

» Packages Developer Builds Java™ 8 Support

Security release: A vulnerability with **JGit client (CVE-2014-9390)** was found, please **check for updates** or download an updated package below.

Mac OS X users please note: Eclipse requires Mac OS X 10.5 (Leopard) or greater. You might get a warning on some installations due to recent changes how Apple treats signatures. Please see **this post** for a possible solution.

Eclipse Luna SR1a (4.4.1) Release for

Windows
Linux
✓ Mac OS X (Cocoa)

Mac OS X users please note: Eclipse requires Mac OS X 10.5 (Leopard) or greater.

Eclipse IDE for Java Developers, 153 MB
Downloaded 1,663,315 Times

The essential tools for any Java developer, including a Java IDE, a CVS client, Git client, XML Editor, Mylyn, Maven integration...

Mac OS X 32 Bit
Mac OS X 64 Bit

Package Solutions Filter Packages

DevOps Services

Develop in the cloud on IBM Bluemix using Eclipse or Orion

eclipsecon 2015
March 9 - 12, 2015 Register Now

RELATED LINKS

- Compare & Combine



From C to Java: Getting Started (1)

Exercise #1

Instructions:

- Unpack and Copy Eclipse to your favourite destination
- Start-up Eclipse



From C to Java: Getting Started (1)

Exercise #1

- Setup your Development Environment (5min)
- **Create your first Project (2min)**
- Create and run your main() method (2min)

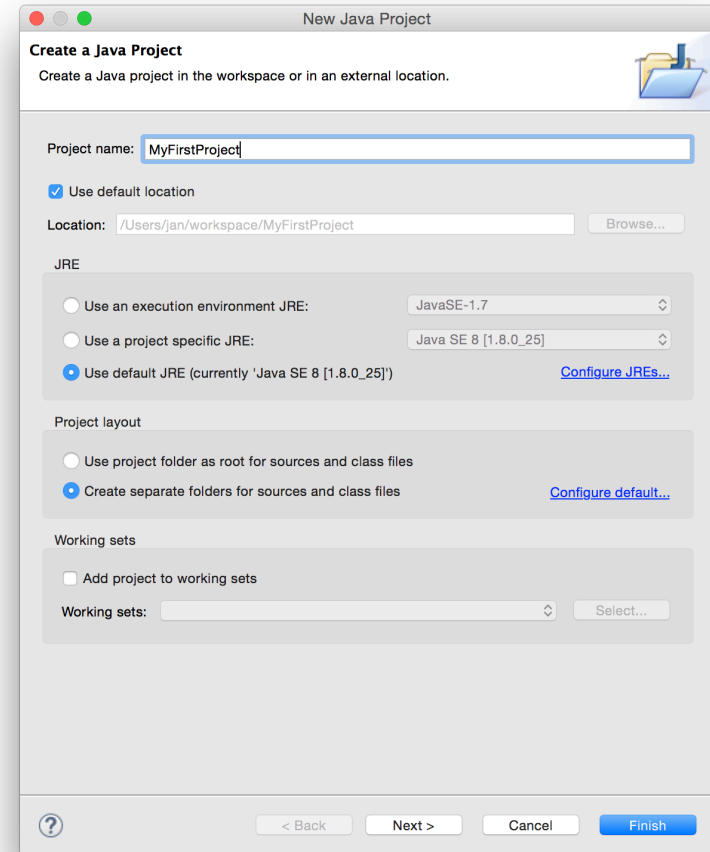


From C to Java: Getting Started (1)

Exercise #1

Instructions:

- Create a new Project
(File – New – Java Project)
- Name your Project:
“MyFirstProject”

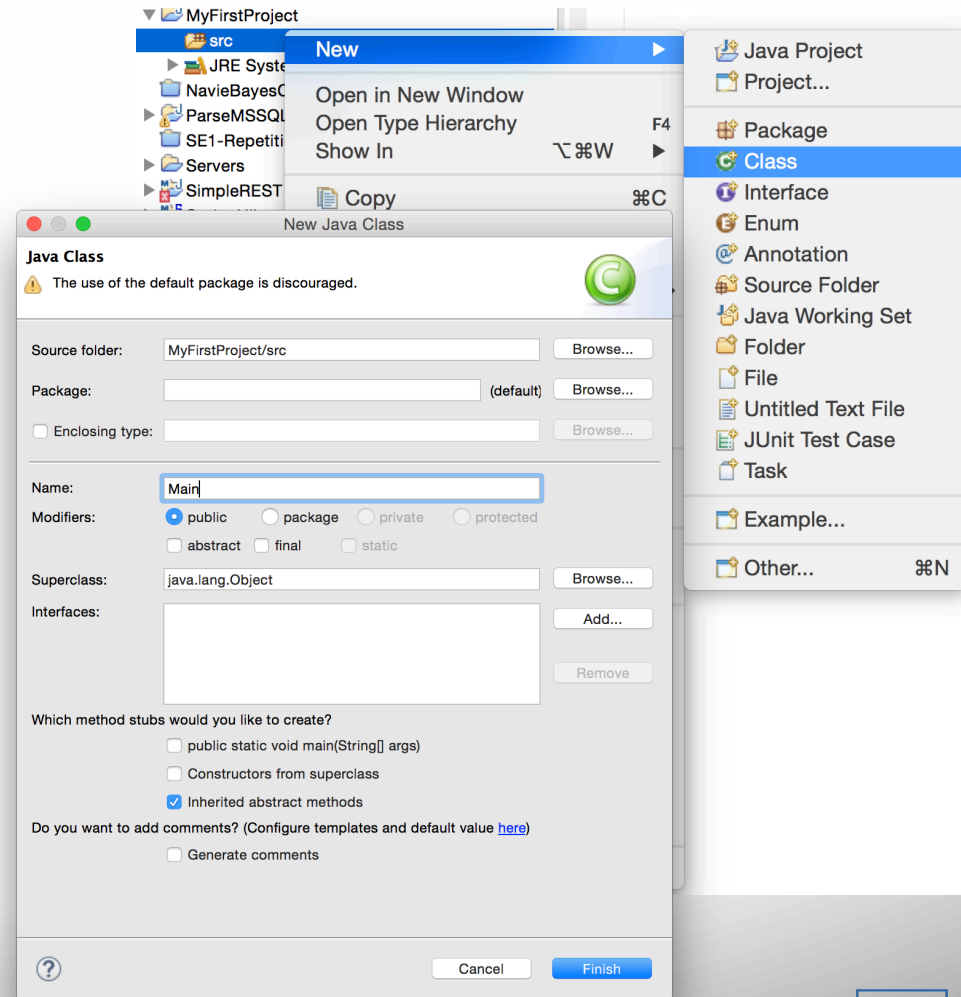


From C to Java: Getting Started (1)

Exercise #1

Instructions:

- Create a Class
(File – New – Class)
- Name it “Main”



From C to Java: Getting Started (1)

Exercise #1

- Setup your Development Environment (5min)
- Create your first Project (2min)
- **Create and run your main() method (2min)**



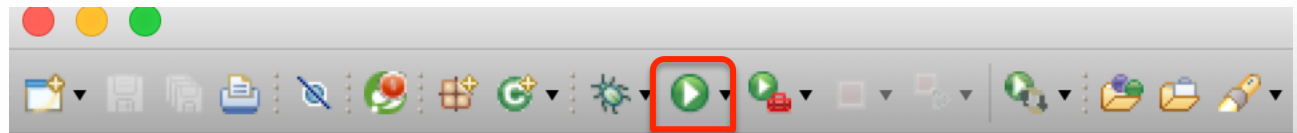
From C to Java: Getting Started (1)

Exercise #1

Instructions:

- Write your main Method
- Click the green “Play” Icon in the Menu bar of eclipse

```
public class Main {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

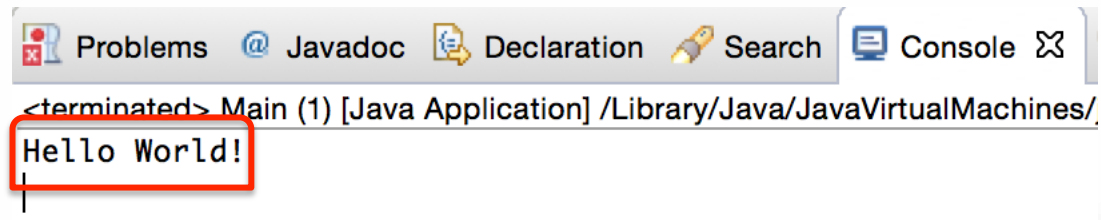


From C to Java: Getting Started (1)

Exercise #1

Instructions:

The following Output should appear in the console:



The screenshot shows a Java IDE's console window. The title bar includes tabs for 'Problems', 'Javadoc', 'Declaration', 'Search', and 'Console'. The console output shows '<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/' followed by 'Hello World!' on a new line. The text 'Hello World!' is enclosed in a red rectangular box.

From C to Java: Getting Started (1)

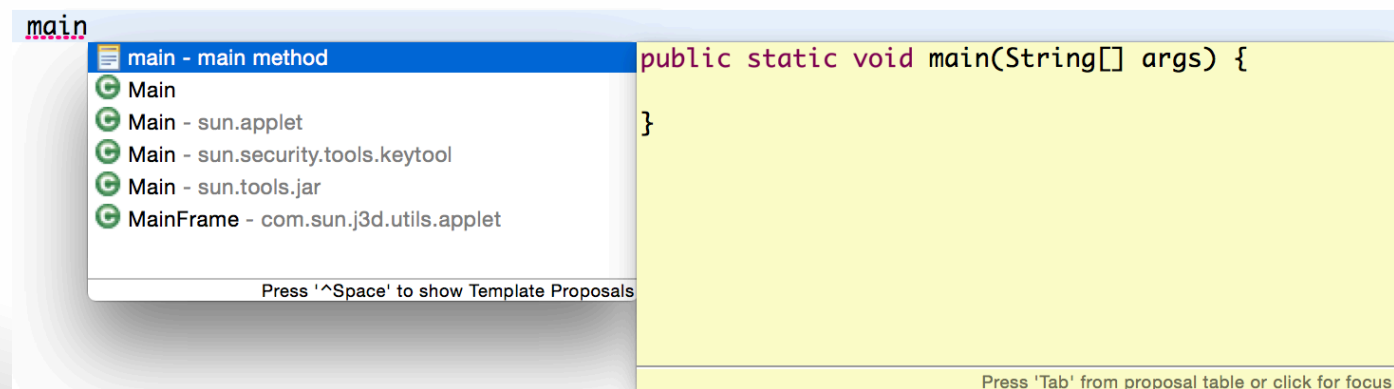
Integrated Development Environment

Auto-complete is your best friend ! (Control-Space)

Well not in the exam. ;-)

Remove your main method and try typing:

“main” and press Control - Space



Session Outline

Session #1

- Getting Started (1)
 - Hello World
 - Primitive Data types
- Getting Started (2)
 - Simple Input / Output
 - Control Structures
- Getting Started (3)
 - Classes in Java, Initialization, Class vs. Instance
 - Information Hiding (public, private, protected)

Session #2

- Getting Started (4)
 - Built-In Classes
 - Type Casting vs. Generics
- Getting Started (5)
 - Java's Object Hierarchy
 - Inheritance vs. Polymorphism
 - Abstract Classes / Interfaces
- Getting Started (6)
 - Parameter Passing
 - Exception Handling



Session #1 – Detailed Outline

Getting Started (2)

- Simple Input, Output
- Control Structures
- Exercise #2:
 - Read keyboard input and write it to the Console
 - Write a simple countdown using:
 - While
 - For
 - Recursion defining your own method (use the static modifier)



From C to Java: Getting Started (2)

Simple Input / Output

You already saw an Output of our “Hello world” program.

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

The System Class Library provides you with Input and Output capabilities by using System.in(..) or System.out(..)



From C to Java: Getting Started (2)

Simple Input / Output

There are different printers available, `printf()` is close to the `printf()` function in C.

```
System.out.print("Prints text without linebreak");  
System.out.println("Prints text with a linebreak at the end!");  
System.out.printf("String: %s \nFloat: %f \nInteger: %d\n\n", "String", 1.0, 1);
```

Since Java SE5 there is a new Class called `Scanner` which eases reading Inputs from a keyboard



From C to Java: Getting Started (2)

Simple Input / Output

Scanner Example:

```
import java.util.Scanner;
public class Main {

    public static void main(String[] args) {
        System.out.println("Write your name and press Enter");
        Scanner readData = new Scanner(System.in);
        System.out.println("Hello, " + readData.next() + "!");
        readData.close();
    }
}
```



From C to Java: Getting Started (2)

Control Structures

For Statement:

Parameters:

(counter init; counter limit; counter increase)

```
for (int i=0; i<10; i++){  
    // do something  
}
```

While Statement:

Parameters:

(counter limit)

```
int c = 10;  
while (c >= 0) {  
    // do something  
}
```



From C to Java: Getting Started (2)

Control Structures

Switch Statement:

Parameters:

switch(expression), case(constant)

```
int dice=5;
switch (dice) {
    case 1:
        //if dice is rolled 1
        break;
    case 2:
        //if dice is rolled 2
        break;
    default:
        break;
}
```

Command Statement:

Parameters:

while(expression)

```
int counter = 10;
do {
    counter--;
}while(counter > 0);
```



From C to Java: Getting Started (2)

Exercise #2

- Read keyboard input and write it to the Console
- Write a simple countdown using: while, for, recursion



From C to Java: Getting Started (2)

Exercise #2

- **Read keyboard input and write it to the Console (5min)**
- Write a simple countdown using: while, for, recursion (5min)



From C to Java: Getting Started (2)

Exercise #2

Instructions:

- Use your existing Main Class and main() method to read from the console using the Scanner class and System.in
- Print the Input back to the Console.



From C to Java: Getting Started (2)

Exercise #2 - Solution

Read keyboard input and write it to the Console

```
public static void main(String[] args) {  
    System.out.println("Enter your name and press Enter:");  
    Scanner s = new Scanner(System.in);  
    System.out.println("Hello, " + s.nextLine() + "!");  
}
```

Enter your name and press Enter:

Jan

Hello, Jan!



From C to Java: Getting Started (2)

Exercise #2

- Read keyboard input and write it to the Console
- **Write a simple countdown using: while, for, recursion (5min)**



From C to Java: Getting Started (2)

Exercise #2

- Recall the structure of for- and while-statements
- Write a countdown which counts down to 0 using System.out.() methods

```
for (int i=0; i<10; i++){  
    // do something  
}
```

```
int c = 10;  
while (c >= 0) {  
    // do something  
}
```


From C to Java: Getting Started (2)

Exercise #2 - Solution

Write a simple countdown using: while, for, recursion

```
for (int i=10; i>=0; i--){  
    System.out.println(i);  
}
```

```
int i = 10;  
while (i >= 0) {  
    System.out.println(i--);  
}
```

```
public static int recursiveCounter(int i) {  
    if (i < 0) {  
        return 0;  
    }  
    System.out.println(i);  
    return recursiveCounter(--i);  
}
```



Session Outline

Session #1

- Getting Started (1)
 - Hello World
 - Primitive Data types
- Getting Started (2)
 - Simple Input / Output
 - Control Structures
- Getting Started (3)
 - Classes in Java, Initialization, Class vs. Instance
 - Information Hiding (public, private, protected)

Session #2

- Getting Started (4)
 - Built-In Classes
 - Type Casting vs. Generics
- Getting Started (5)
 - Java's Object Hierarchy
 - Inheritance vs. Polymorphism
 - Abstract Classes / Interfaces
- Getting Started (6)
 - Parameter Passing
 - Exception Handling



Session #1 – Detailed Outline

Getting Started (3)

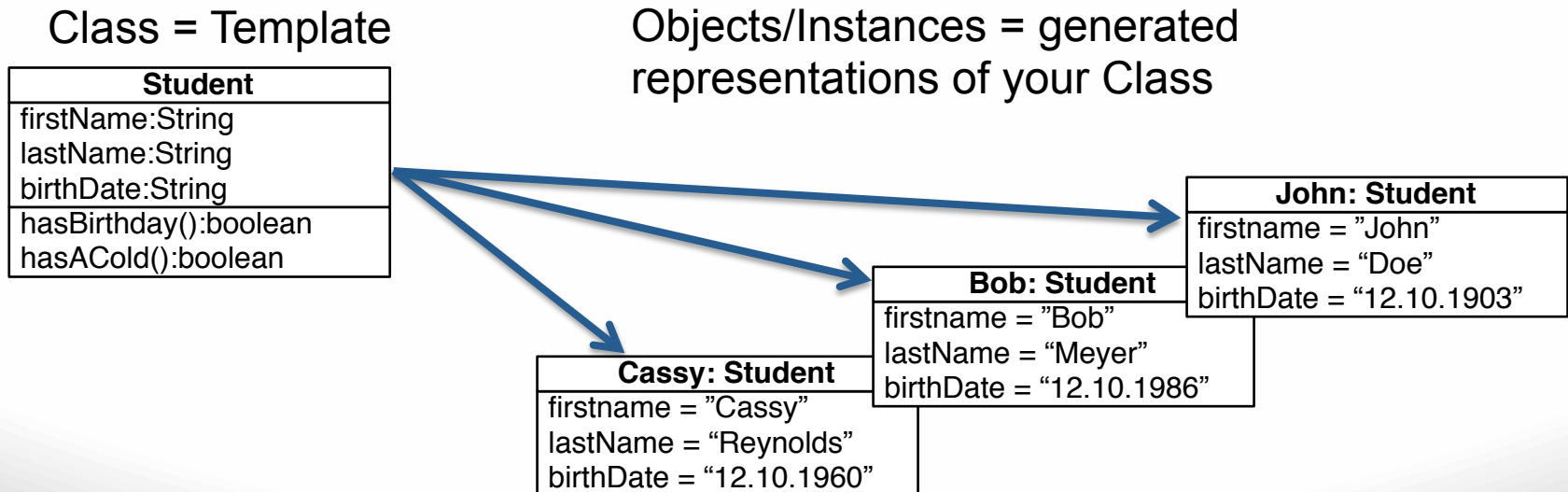
- Classes in Java, Initialization, Class vs. Instance
- Information Hiding (public, private, protected)
- Exercise #3:
 - Write your own class (Student)
 - Add Constructor to your class
 - Make sure a birthdate can not be changed after creation



From C to Java: Getting Started (3)

Classes in Java

What is a Class? – It's a template to describe your own or existing complex data types



From C to Java: Getting Started (3)

Classes in Java

Define a Class

```
public class Student {  
  
    String firstName;  
    String lastName;  
    String birthDate;  
}
```

Create an Instance/Object

In your main() method:

```
Student bob = new Student();
```



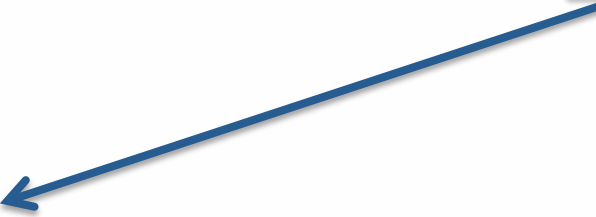
From C to Java: Getting Started (3)

Classes in Java

Object initialization by using a „**Constructor**“
The default Constructor is available implicitly

```
public class Student {  
    String firstName;  
    String lastName;  
    String birthDate;  
  
    public Student() {  
        //do something in here..  
    }  
}
```

Student bob = new Student();



So everything is fine? – What do we miss?

From C to Java: Getting Started (3)

Classes in Java

Drawback of using the default Constructor:
Attribute values are not assigned during Instance/Object creation

Example:

```
...  
Student bob = new Student();  
System.out.println(bob.firstName);  
// -> returns „null“  
...
```

Hint: Access class attributes by using the dot-Notation.
(className.attributeName)



From C to Java: Getting Started (3)

Classes in Java

The Solution: Write your own Constructor

```
public class Student {  
    String firstName;  
    String lastName;  
    String birthDate;  
  
    public Student(String firstName, String lastName, String birthDate)  
    {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.birthDate = birthDate;  
    }  
}
```

class attributes method parameters

What is “this”? – Why do we need it?



From C to Java: Getting Started (3)

Information Hiding

Information Hiding is a very important principle in Object Oriented Programming:

Modifiers:

public:	access from every class
private:	no access outside class
protected:	access from specialisations
package:	access inside a package
default:	access inside a package



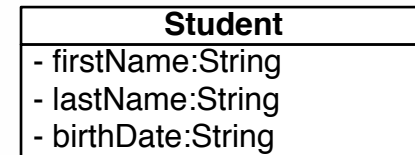
From C to Java: Getting Started (3)

Information Hiding

Example Code:

```
public class Student {  
    private String firstName;  
    private String lastName;  
    private String birthDate;  
}
```

Corresponding Diagram:



From C to Java: Getting Started (3)

Information Hiding

Access methods (Getter/Setter) help to access your Attributes even if they are private:

```
public class Student {  
    private String firstName;  
    private String lastName;  
    private String birthDate;  
  
    public String getFirstName() {  
        return firstName;  
    }  
    public void setFirstName(String firstName) {  
        this.firstName = firstName;  
    }  
    ...  
}
```

Student
- firstName:String - lastName:String - birthDate:String
+ getFirstName():String + setFirstName(firstName:String):void + getLastName():String + setLastName(lastName:String): void + getBirthDate():String + setBirthDate(birthDate:String) : void



From C to Java: Getting Started (3)

Information Hiding

Access methods vs. public parameters:

Access methods are more fine granular according to read and write access

e.g.

read firstName

read/write lastName

read birthDate

Student	
- firstName:String	
- lastName:String	
- birthDate:String	
+ getFirstName():String	
+ getLastName():String	
+ setLastName(lastName:String): void	
+ getBirthDate():String	



From C to Java: Getting Started (3)

Exercise #3

- Write your own Class (Student)
- Add a Constructor to your Class
- Make sure the “birthDate” parameter can not be changed after the Student Instance has been created.



From C to Java: Getting Started (3)

Exercise #3

- **Write your own Class (Student) (2min)**
- Add a Constructor to your Class (2min)
- Make sure the “birthDate” parameter can not be changed after the Student Instance has been created. (3min)

Refer to the following model to create your Student class:

Student
firstName:String
lastName:String
birthDate:String



From C to Java: Getting Started (3)

Exercise #3 - Solution

Write your own Class (Student)

```
public class Student {  
    String firstName;  
    String lastName;  
    String birthDate;  
}
```



From C to Java: Getting Started (3)

Exercise #3

- Write your own Class (Student) (2min)
- **Add a Constructor to your Class (2 min)**
- Make sure the “birthDate” parameter can not be changed after the Student Instance has been created.(3min)



From C to Java: Getting Started (3)

Exercise #3 - Solution

Add a Constructor to your Class

```
public class Student {  
    String firstName;  
    String lastName;  
    String birthDate;  
  
    public Student(String firstName, String lastName, String birthDate) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.birthDate = birthDate;  
    }  
}
```

class attributes method parameters



From C to Java: Getting Started (3)

Exercise #3

- Write your own Class (Student) (2min)
- Add a Constructor to your Class (2min)
- **Make sure the “birthDate” parameter can not be changed after the Student Instance has been created. (3 min)**

Hint: Use access methods (getter and setter) to implement different access



From C to Java: Getting Started (3)

Exercise #3 – Solution

Make sure the “birthDate” parameter can not be changed after the Student Instance has been created.

```
public class Student {
    String firstName;
    private String lastName;
    String birthDate;

    public Student(String firstName, String lastName, String birthDate) {

        this.firstName = firstName;
        this.lastName = lastName;
        this.birthDate = birthDate;
    }

    public String getBirthDate() { // birthDate read access
        return birthDate;
    }
    // getters and setters for firstName and lastName
}
```



Session Outline

Session #1

- Getting Started (1)
 - Hello World
 - Primitive Data types
- Getting Started (2)
 - Simple Input / Output
 - Control Structures
- Getting Started (3)
 - Classes in Java, Initialization, Class vs. Instance
 - Information Hiding (public, private, protected)

Session #2

- Getting Started (4)
 - Built-In Classes
 - Type Casting vs. Generics
- Getting Started (5)
 - Java's Object Hierarchy
 - Inheritance vs. Polymorphism
 - Abstract Classes / Interfaces
- Getting Started (6)
 - Parameter Passing
 - Exception Handling



Session #2 – Detailed Outline

Getting started (4)

- Built-In Classes:
 - LinkedList, HashMaps, Stacks, Queues
- Type Casting vs. Generics
 - Runtime errors vs. Compile time errors
- Exercise #4:
 - Write a LinkedList for your own Class (Student)
 - Add Elements to LinkedList and Iterate over them to print out each firstName



From C to Java: Getting Started (4)

Build-in Classes

In C you already learned how to write a Queue and Stack. Java provides various different build-in classes for reuse:

<code>java.util.Stack,</code>	methods: <code>push()</code> , <code>pop()</code> , <code>peek()</code>
<code>java.util.Queue,</code>	methods: <code>poll()</code> , <code>add()</code> , <code>peek()</code>
<code>java.util.LinkedList,</code>	methods: <code>add()</code> , <code>remove()</code>
<code>java.util.HashMap,</code>	methods: <code>put(key,value)</code> , <code>remove(key)</code>
...	

From C to Java: Getting Started (4)

Build-in Classes

Use of existing Classes:

1. Import the needed Classes
2. Initialize the Class using the given Constructor
3. Call methods on the Class using the dot operator



From C to Java: Getting Started (4)

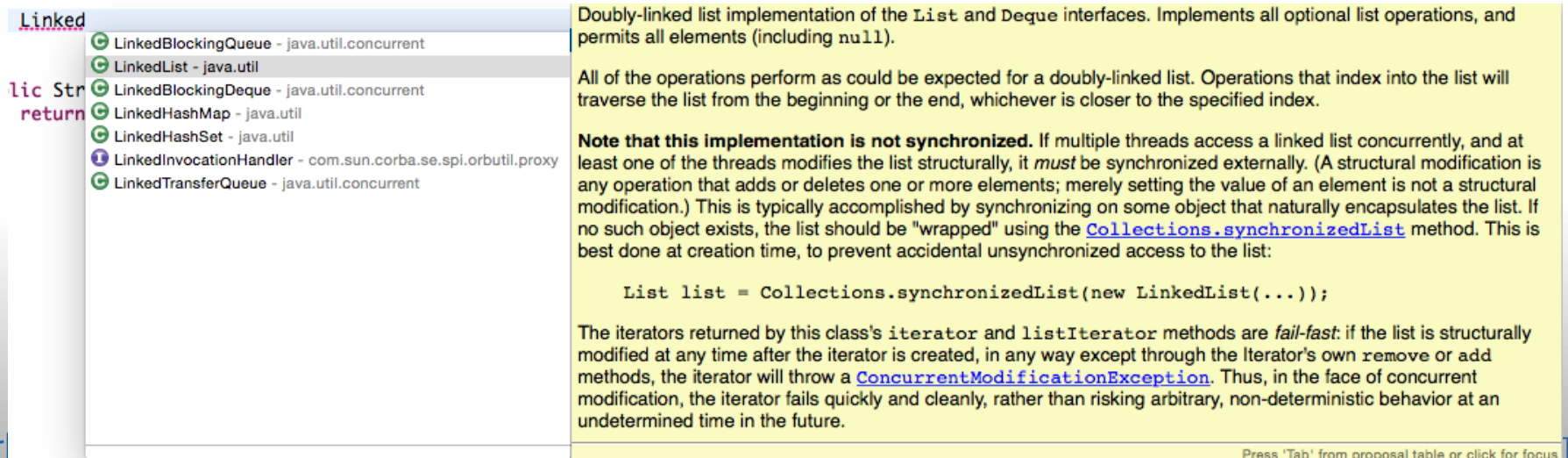
Build-in Classes

Example: Implementing a LinkedList:

Start typing: “Linked” use Ctrl + Space

```
import java.util.LinkedList;
```

```
public static void main(String[] args) {  
    LinkedList<String> stringList = new LinkedList<String>();  
}
```



The screenshot shows an IDE with a code completion menu open for the word "Linked". The menu lists several classes, with "LinkedList - java.util" selected. To the right of the menu, a tooltip for the "LinkedList" class is displayed. The tooltip contains the following text:

Doubly-linked list implementation of the `List` and `Deque` interfaces. Implements all optional list operations, and permits all elements (including `null`).

All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

Note that this implementation is not synchronized. If multiple threads access a linked list concurrently, and at least one of the threads modifies the list structurally, it *must* be synchronized externally. (A structural modification is any operation that adds or deletes one or more elements; merely setting the value of an element is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the list. If no such object exists, the list should be "wrapped" using the [Collections.synchronizedList](#) method. This is best done at creation time, to prevent accidental unsynchronized access to the list:

```
List list = Collections.synchronizedList(new LinkedList(...));
```

The iterators returned by this class's `iterator` and `listIterator` methods are *fail-fast*: if the list is structurally modified at any time after the iterator is created, in any way except through the iterator's own `remove` or `add` methods, the iterator will throw a [ConcurrentModificationException](#). Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Press 'Tab' from proposal table or click for focus

From C to Java: Getting Started (4)

Type Casting vs. Generics

In Java typed parameters are called Generics:

Typed Parameter: A restriction onto a Parameter which only allows a subset of accepted class types.

Example:

A list which only allows Student Objects to be added.

```
LinkedList<String> stringList = new LinkedList<String>();
```



From C to Java: Getting Started (4)

Type Casting vs. Generics

Type-Cast Example:

What will happen if you try compile and run this code?

```
LinkedList genericList = new LinkedList();  
Car car = new Car();  
genericList.add(car);  
Student studentInList = (Student) (genericList.get(0));  
System.out.println(studentInList.firstName);
```

Runtime Exception:

java.lang.ClassCastException: X cannot be cast to Y



From C to Java: Getting Started (4)

Type Casting vs. Generics

Generics Example:

What will happen if you try compile and run this code?

```
LinkedList<Student> studentList = new LinkedList<Student>();  
Car car = new Car();  
studentList.add(car);
```

Compile Time Exception:

java.lang.Error: Unresolved compilation problem



From C to Java: Getting Started (4)

Exercise #4

- Write a LinkedList for your own Class (Student)
- Add Elements to LinkedList and Iterate over them to print out each firstName



From C to Java: Getting Started (4)

Exercise #4

- **Write a LinkedList for your own Class (Student) (2min)**
- Add Elements to LinkedList and Iterate over them to print out each firstName (2min)

Hint: Use <Student> as Generic specification to avoid a typecast



From C to Java: Getting Started (4)

Exercise #4 – Solution

Write a LinkedList for your own Class (Student)

```
public static void main(String[] args) {  
    LinkedList<Student> studentList = new LinkedList<Student>();  
}
```



From C to Java: Getting Started (4)

Exercise #4

- Write a LinkedList for your own Class (Student) (2min)
- **Add Elements to LinkedList and Iterate over them to print out each firstName (2min)**

Hint: Use the for-each construct to easily iterate over your List
for (Student s : studentList) {
 //access student s
}



From C to Java: Getting Started (4)

Exercise #4 – Solution

Add Elements to LinkedList and Iterate over them to print out each firstName

```
public static void main(String[] args) {  
    LinkedList<Student> studentList = new LinkedList<Student>();  
    Student john = new Student("John", "Doe", "12/12/1995");  
    Student james = new Student("James", "Bond", "12/10/1983");  
    Student sam = new Student("Sam", "Smith", "12/08/1863");  
    studentList.add(john);  
    studentList.add(james);  
    studentList.add(sam);  
  
    for (Student s : studentList) {  
        System.out.println(s.firstName);  
    }  
}
```



Session Outline

Session #1

- Getting Started (1)
 - Hello World
 - Primitive Data types
- Getting Started (2)
 - Simple Input / Output
 - Control Structures
- Getting Started (3)
 - Classes in Java, Initialization, Class vs. Instance
 - Information Hiding (public, private, protected)

Session #2

- Getting Started (4)
 - Built-In Classes
 - Type Casting vs. Generics
- Getting Started (5)
 - Java's Object Hierarchy
 - Inheritance vs. Polymorphism
 - Abstract Classes / Interfaces
- Getting Started (6)
 - Parameter Passing
 - Exception Handling



Session #2 – Detailed Outline

Getting started (5)

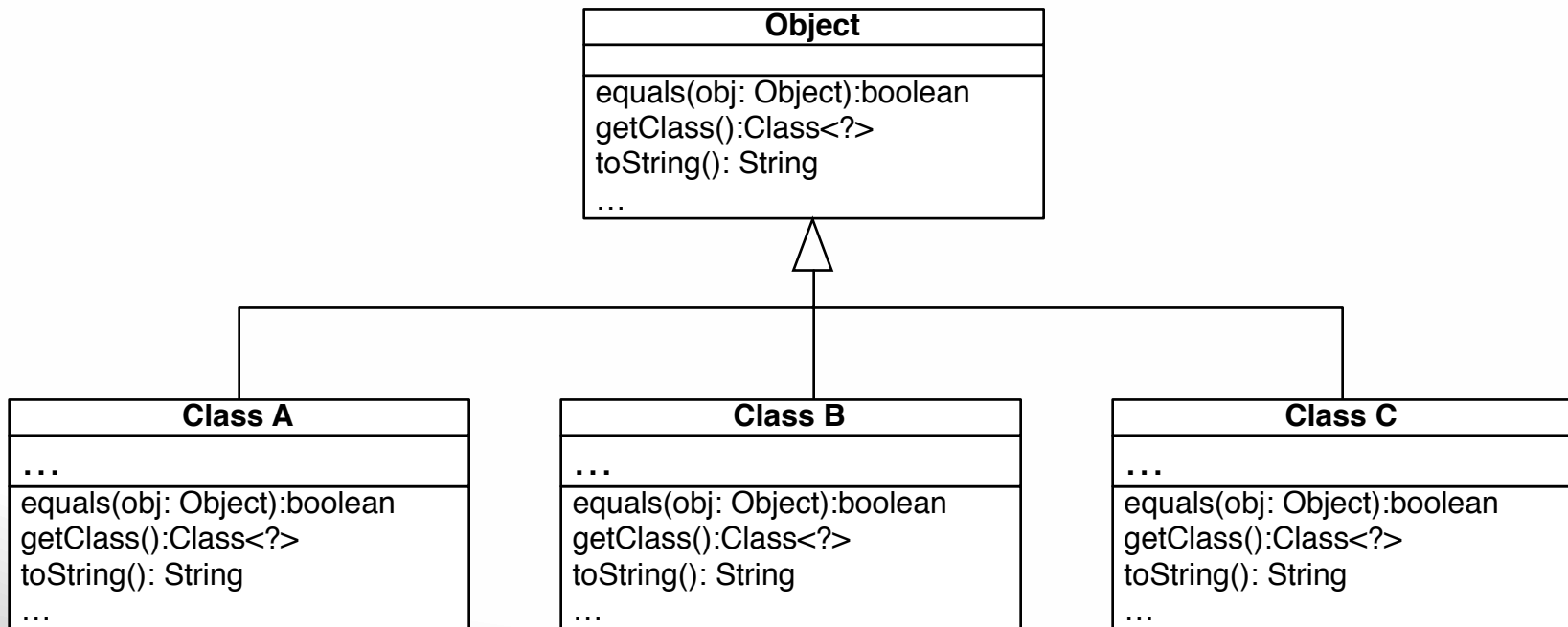
- Java's Object Hierarchy
- Inheritance vs. Polymorphism
- Exercise #5:
 - Create your own class hierarchy for GroundVehicles
 - Override the toString() Method for Car and Motorcycle



From C to Java: Getting Started (5)

Java's Object Hierarchy

In Java almost everything is an Object
So what is not an Object in Java?

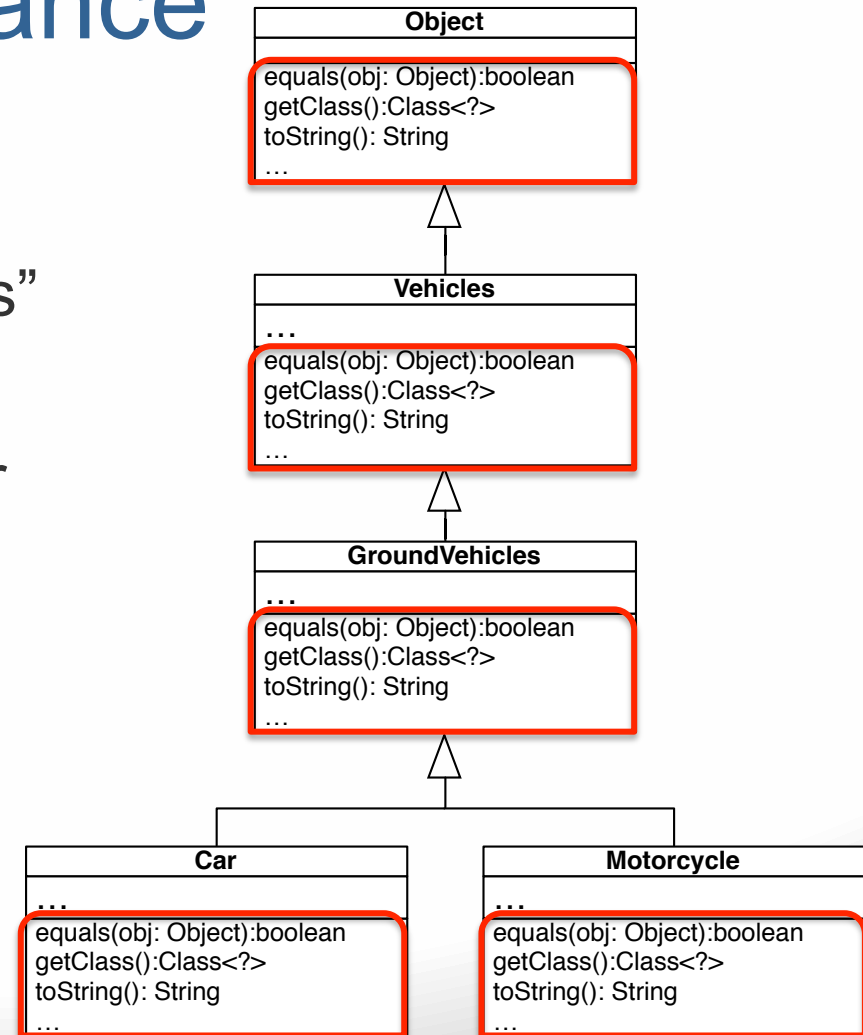


From C to Java: Getting Started (5)

Inheritance

Inheritance allows us to access attributes and methods of the “Superclass” in the “Child classes”

In other words existing behaviour and elements can be reused

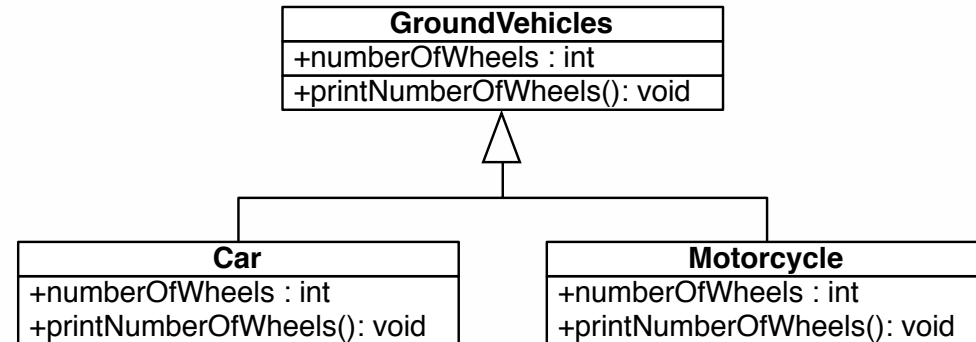


From C to Java: Getting Started (5)

Inheritance

Lets create our own Class hierarchy:

GroundVehicles should be the superclass and Car and Motorcycle should be Child/ Sub-classes



From C to Java: Getting Started (5)

Inheritance

Creating our own hierarchy:

1. Create the Superclass

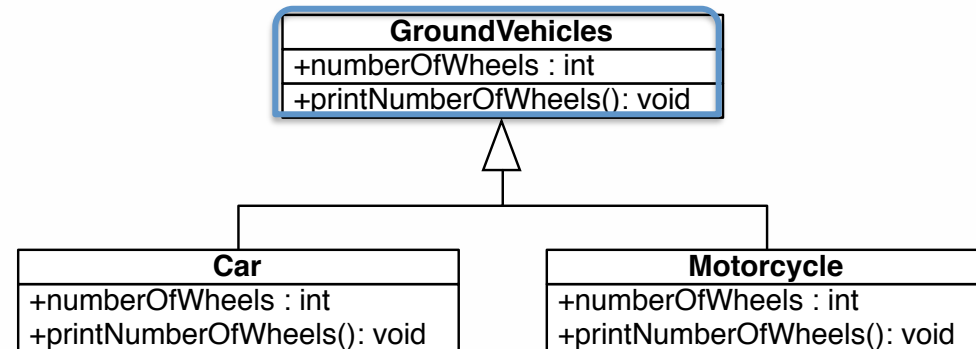
```
public class GroundVehicles {
```

```
    int numberOfWheels;
```

```
    public GroundVehicles(int numberOfWheels) {  
        this.numberOfWheels = numberOfWheels;  
    }
```

```
    public void printNumberOfWheels() {  
        System.out.println("This Vehicle has " + numberOfWheels +  
            "Wheels!");  
    }
```

```
}
```

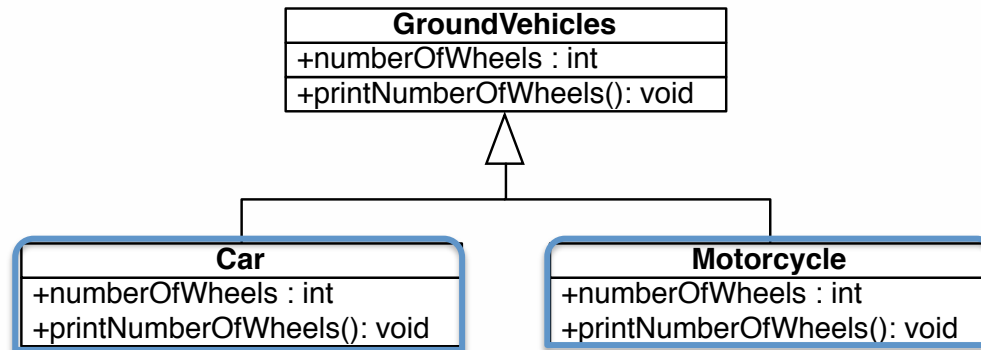


From C to Java: Getting Started (5)

Inheritance

Creating our own hierarchy:

2. Create Child/Sub- Classes



```
public class Car {  
  
}  
  
public class Motorcycle {  
  
}
```

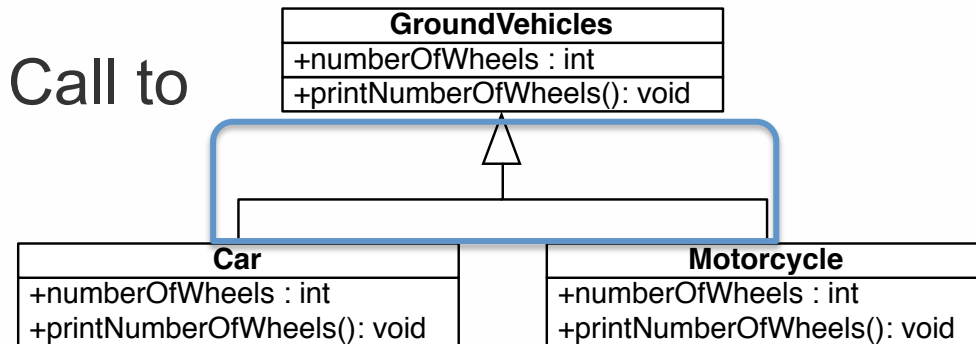
From C to Java: Getting Started (5)

Inheritance

Creating our own hierarchy:

3. Create Relationship using the ***extends*** keyword:

Use a ***super()*** – Constructor Call to pass data to our Superclass



```
public class Car extends GroundVehicles{

    public Car(int numberOfWheels) {
        super(numberOfWheels); //call constructor of GroundVehicles
    }
}
```

