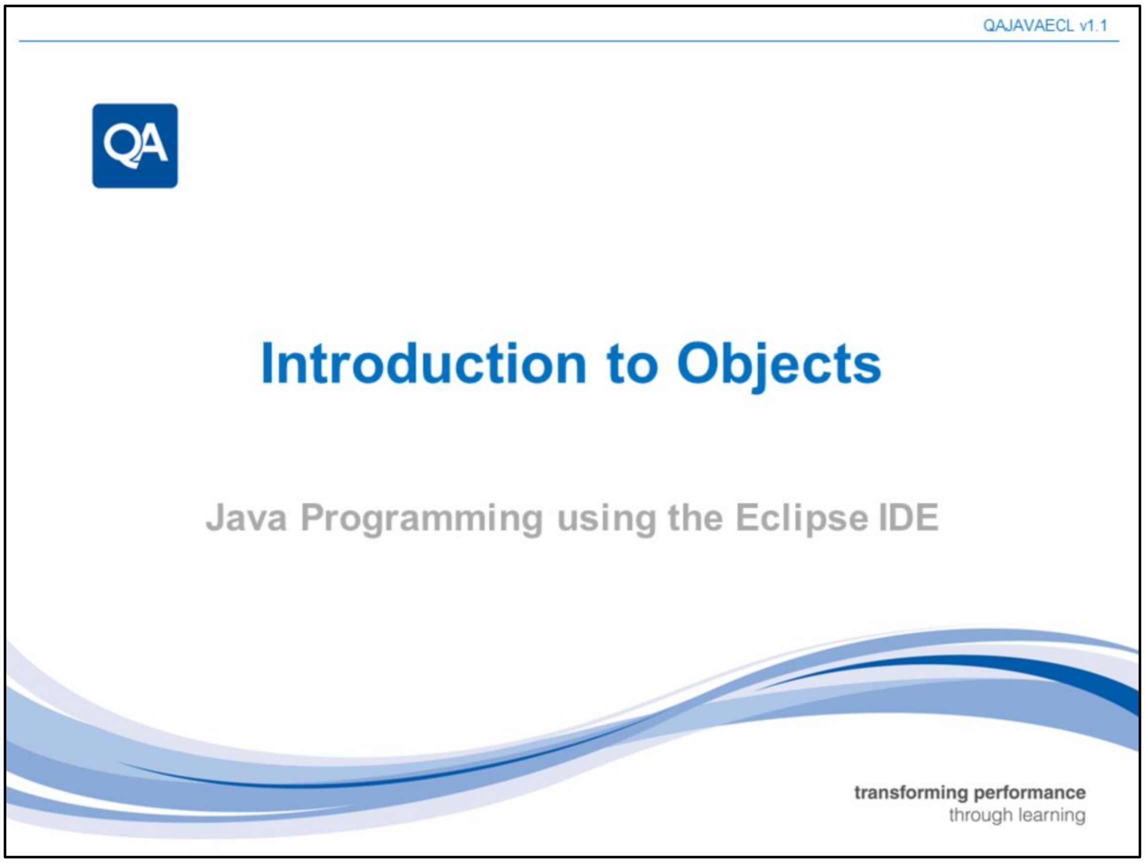
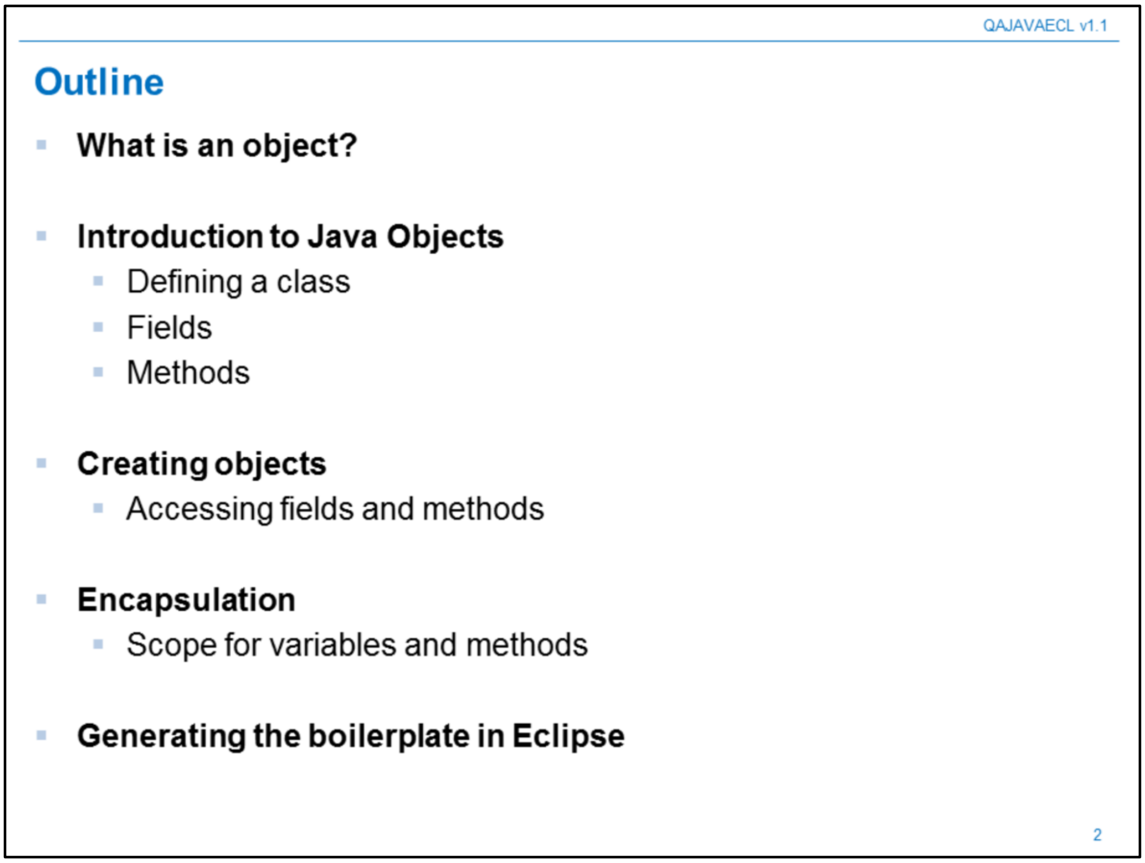


QAJAVAECL v1.1

Introduction to Objects

Java Programming using the Eclipse IDE

transforming performance
through learning

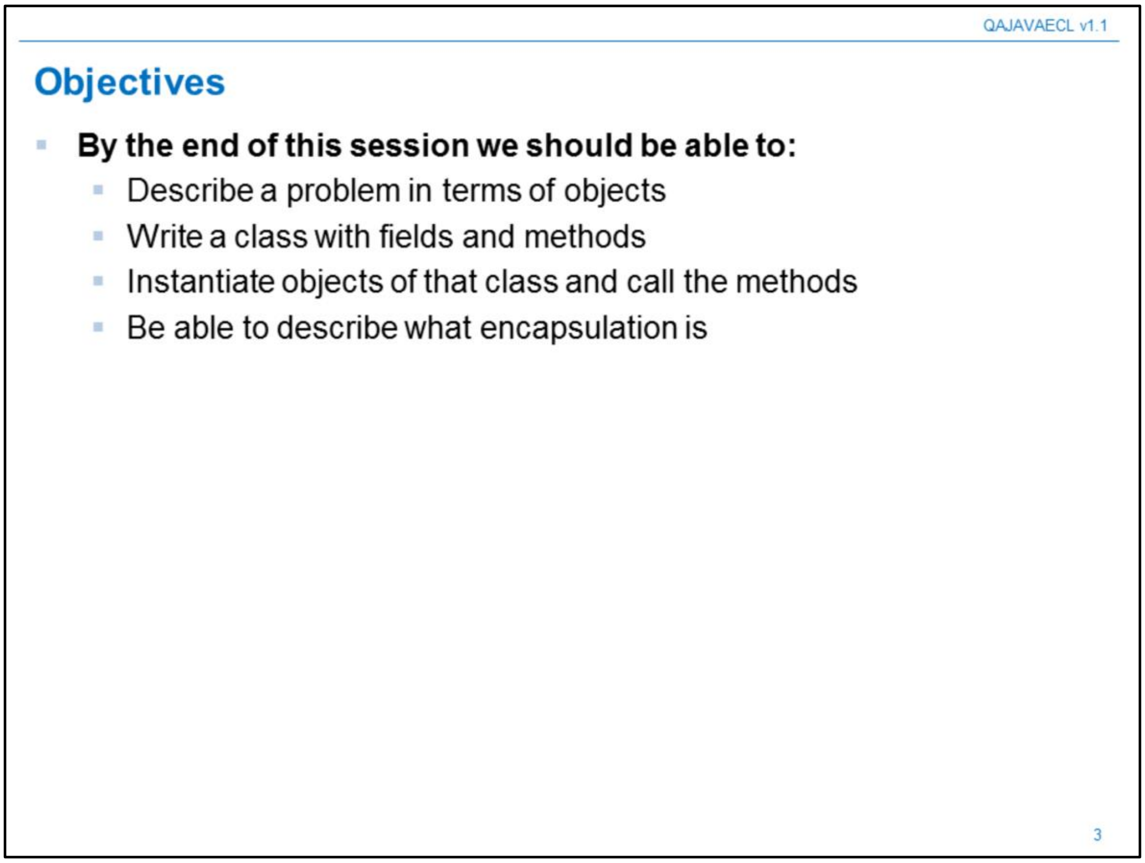


QAJAVAECL v1.1

Outline

- **What is an object?**
- **Introduction to Java Objects**
  - Defining a class
  - Fields
  - Methods
- **Creating objects**
  - Accessing fields and methods
- **Encapsulation**
  - Scope for variables and methods
- **Generating the boilerplate in Eclipse**

2

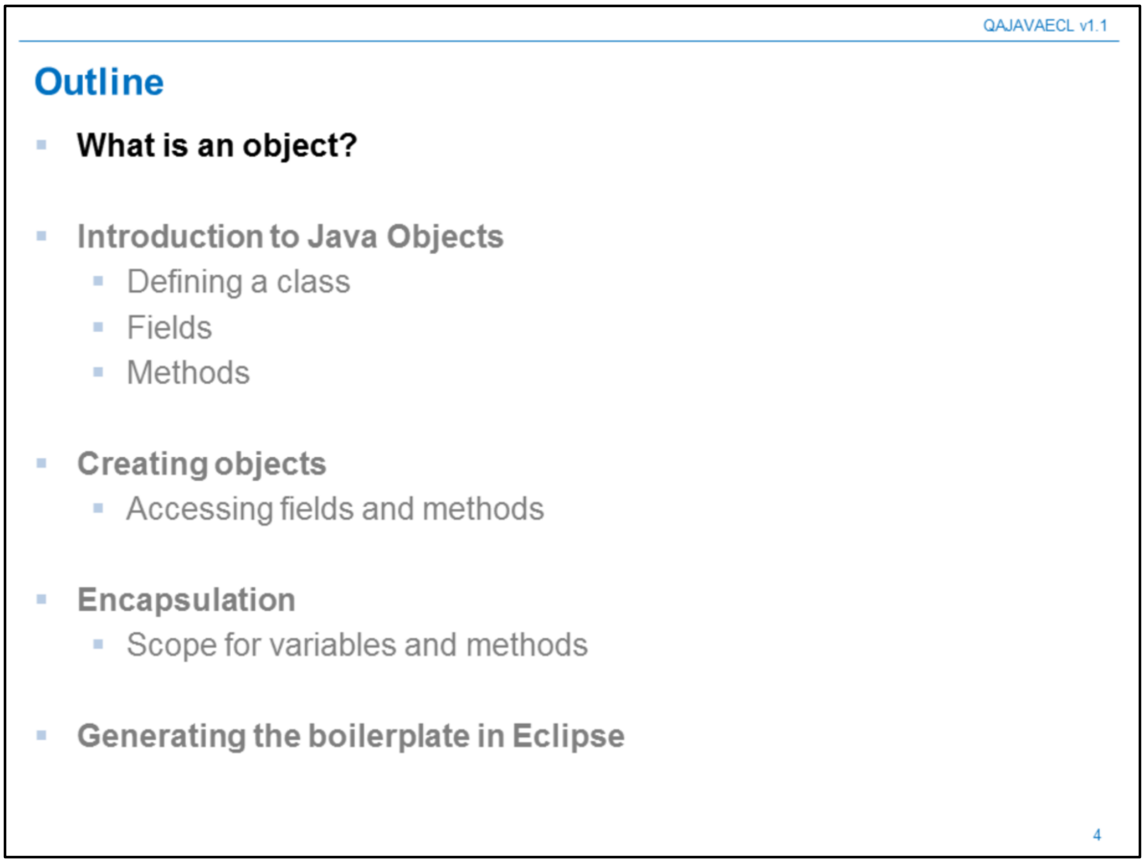


QAJAVAECL v1.1

Objectives

- **By the end of this session we should be able to:**
  - Describe a problem in terms of objects
  - Write a class with fields and methods
  - Instantiate objects of that class and call the methods
  - Be able to describe what encapsulation is

3

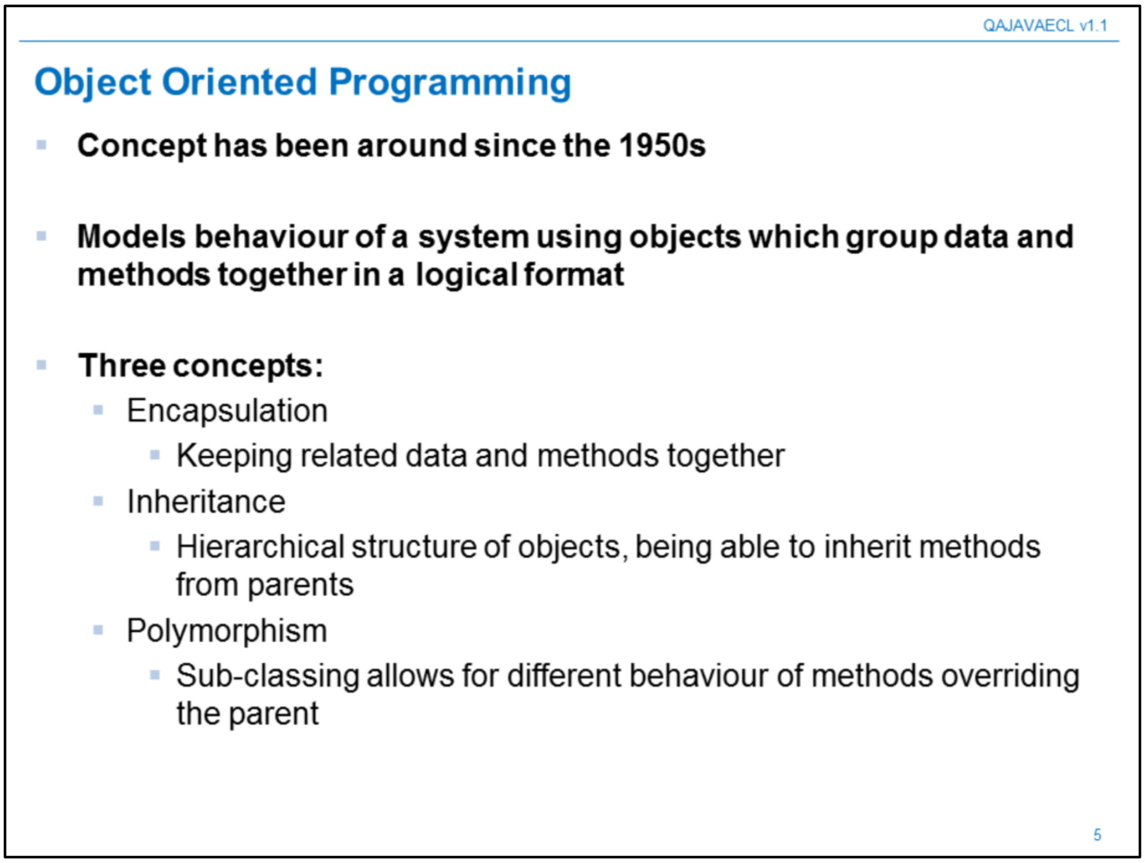


QAJAVAECL v1.1

Outline

- **What is an object?**
- **Introduction to Java Objects**
  - Defining a class
  - Fields
  - Methods
- **Creating objects**
  - Accessing fields and methods
- **Encapsulation**
  - Scope for variables and methods
- **Generating the boilerplate in Eclipse**

4

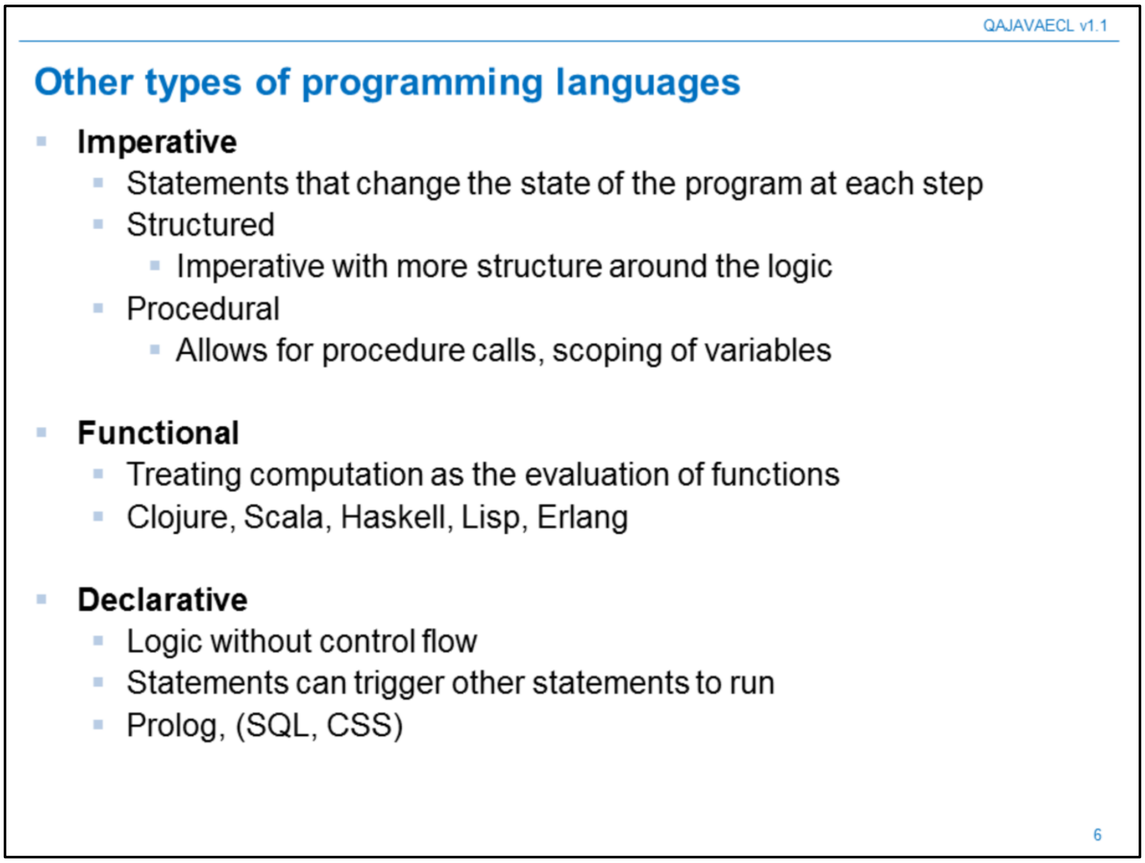


QAJAVAECL v1.1

Object Oriented Programming

- **Concept has been around since the 1950s**
- **Models behaviour of a system using objects which group data and methods together in a logical format**
- **Three concepts:**
  - Encapsulation
    - Keeping related data and methods together
  - Inheritance
    - Hierarchical structure of objects, being able to inherit methods from parents
  - Polymorphism
    - Sub-classing allows for different behaviour of methods overriding the parent

5



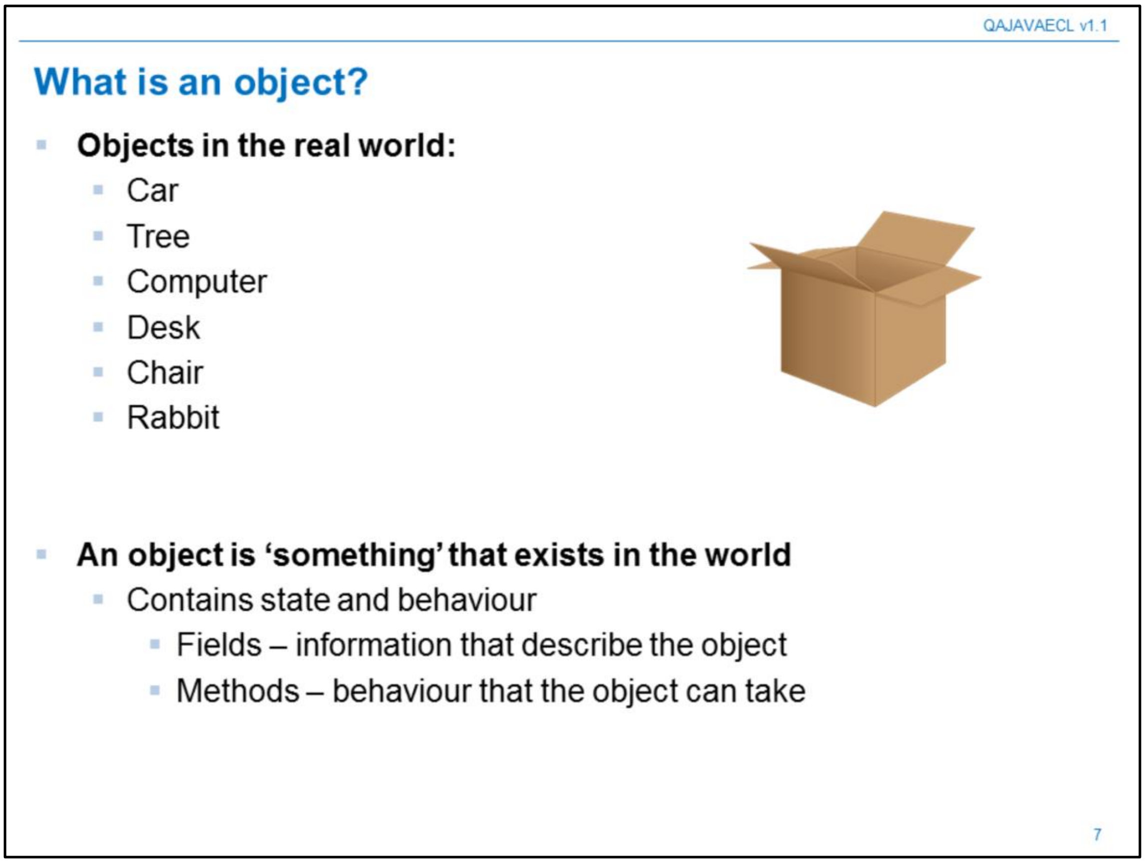
QAJAVAECL v1.1

Other types of programming languages

- **Imperative**
  - Statements that change the state of the program at each step
  - Structured
    - Imperative with more structure around the logic
  - Procedural
    - Allows for procedure calls, scoping of variables
- **Functional**
  - Treating computation as the evaluation of functions
  - Clojure, Scala, Haskell, Lisp, Erlang
- **Declarative**
  - Logic without control flow
  - Statements can trigger other statements to run
  - Prolog, (SQL, CSS)

6

This isn't close to all the examples we can give, or all the paradigms. There are many different ones! A good comparison of some of the language paradigms is at http://en.wikipedia.org/wiki/Comparison_of_programming_paradigms



QAJAVAECL v1.1

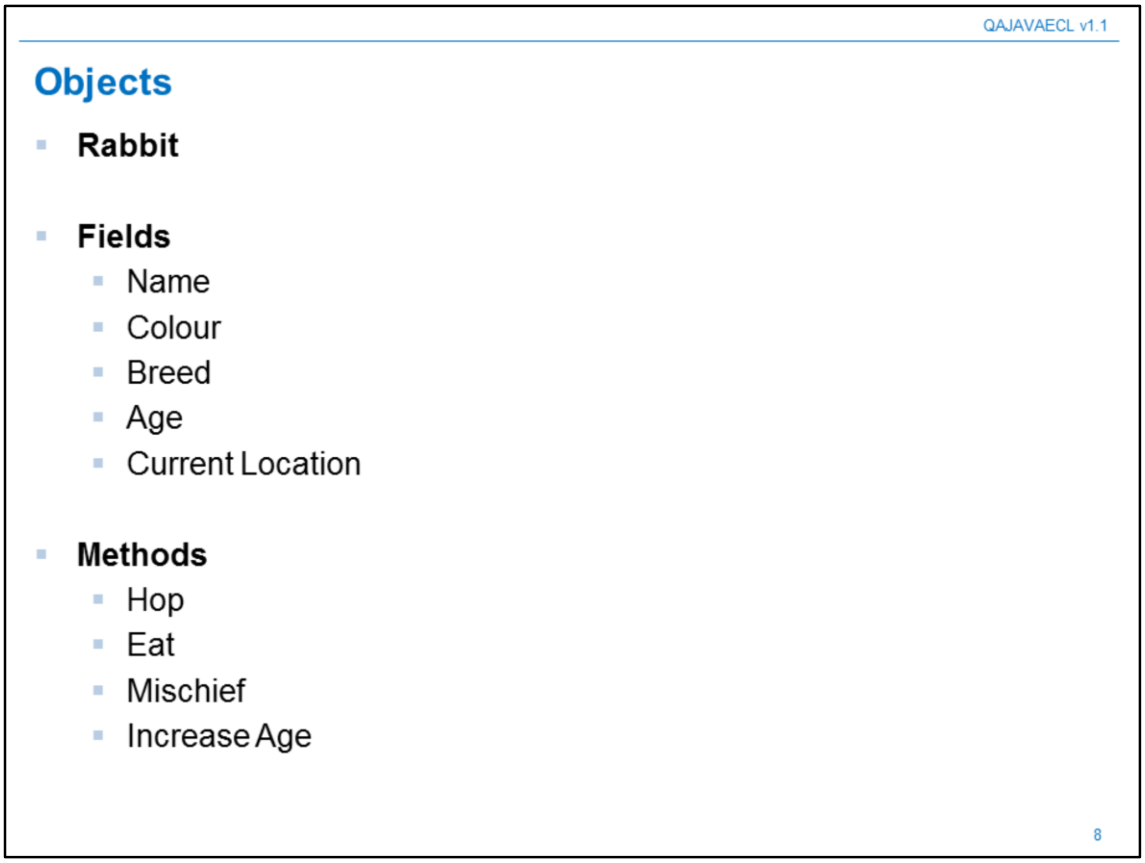
What is an object?

- **Objects in the real world:**
  - Car
  - Tree
  - Computer
  - Desk
  - Chair
  - Rabbit

- **An object is ‘something’ that exists in the world**
  - Contains state and behaviour
    - Fields – information that describe the object
    - Methods – behaviour that the object can take

7

Another way to look at it is that the nouns in a description would be your objects, verbs are methods in the objects, adjectives describe the object and these are the fields. This doesn’t apply for all cases and fields!

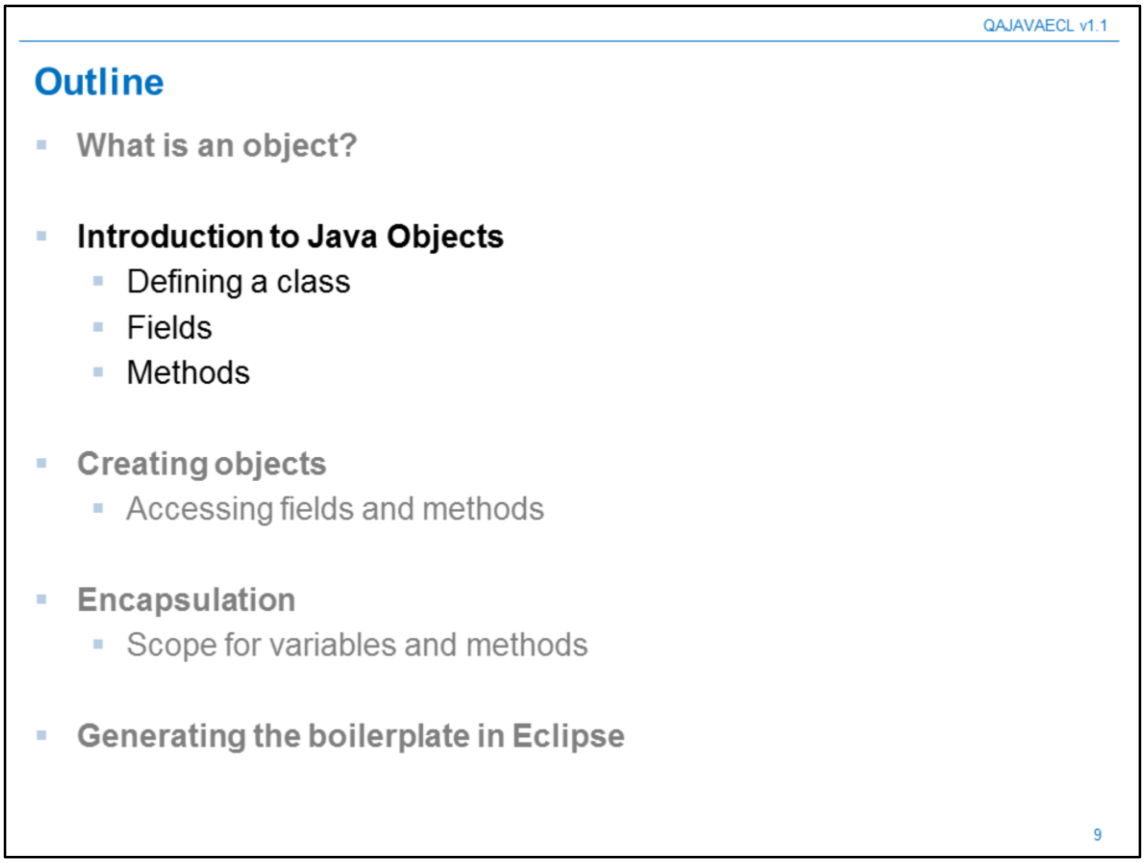


QAJAVAECL v1.1

Objects

- **Rabbit**
- **Fields**
  - Name
  - Colour
  - Breed
  - Age
  - Current Location
- **Methods**
  - Hop
  - Eat
  - Mischief
  - Increase Age

8

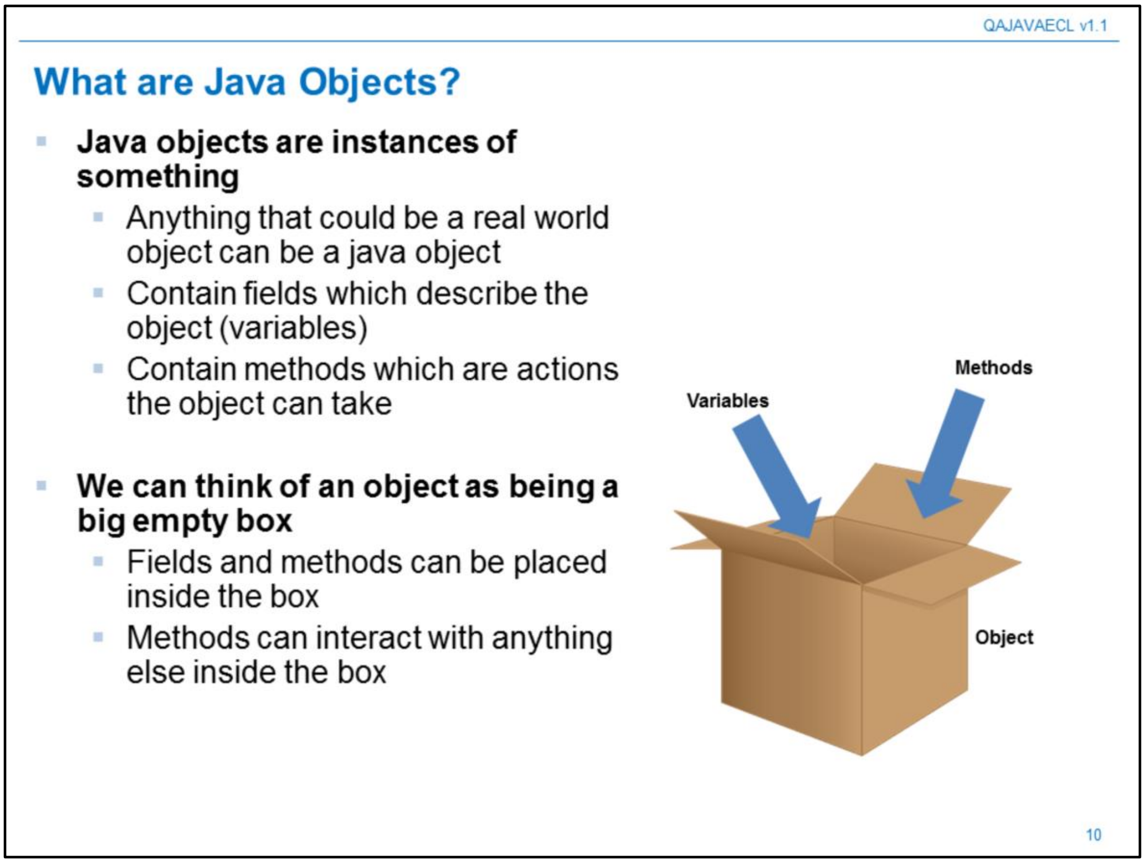


QAJAVAECL v1.1

Outline

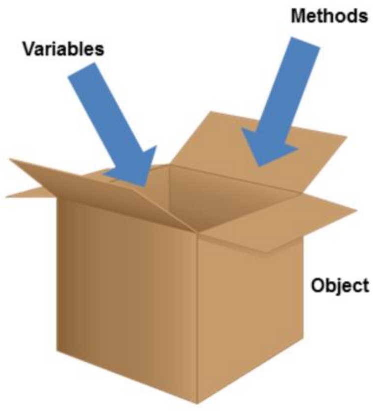
- What is an object?
- **Introduction to Java Objects**
  - Defining a class
  - Fields
  - Methods
- Creating objects
  - Accessing fields and methods
- Encapsulation
  - Scope for variables and methods
- Generating the boilerplate in Eclipse

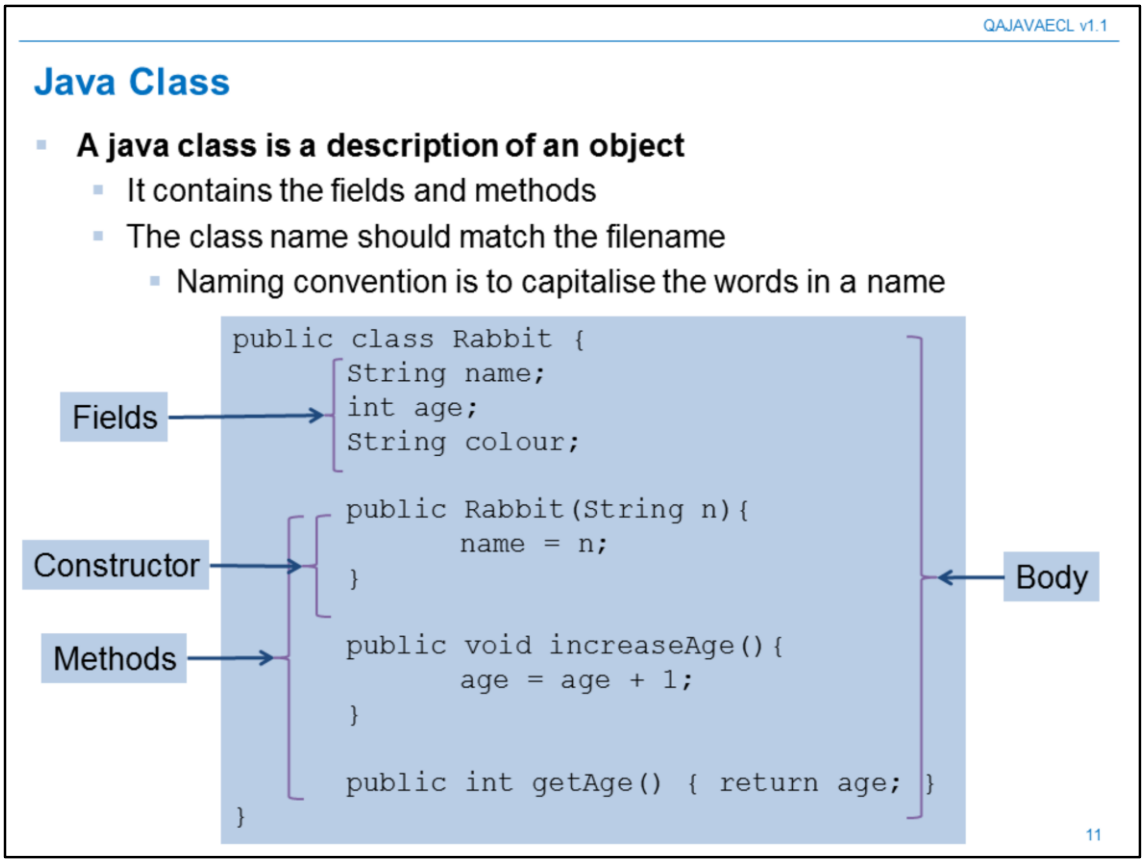
9



What are Java Objects?

- **Java objects are instances of something**
  - Anything that could be a real world object can be a java object
  - Contain fields which describe the object (variables)
  - Contain methods which are actions the object can take
- **We can think of an object as being a big empty box**
  - Fields and methods can be placed inside the box
  - Methods can interact with anything else inside the box



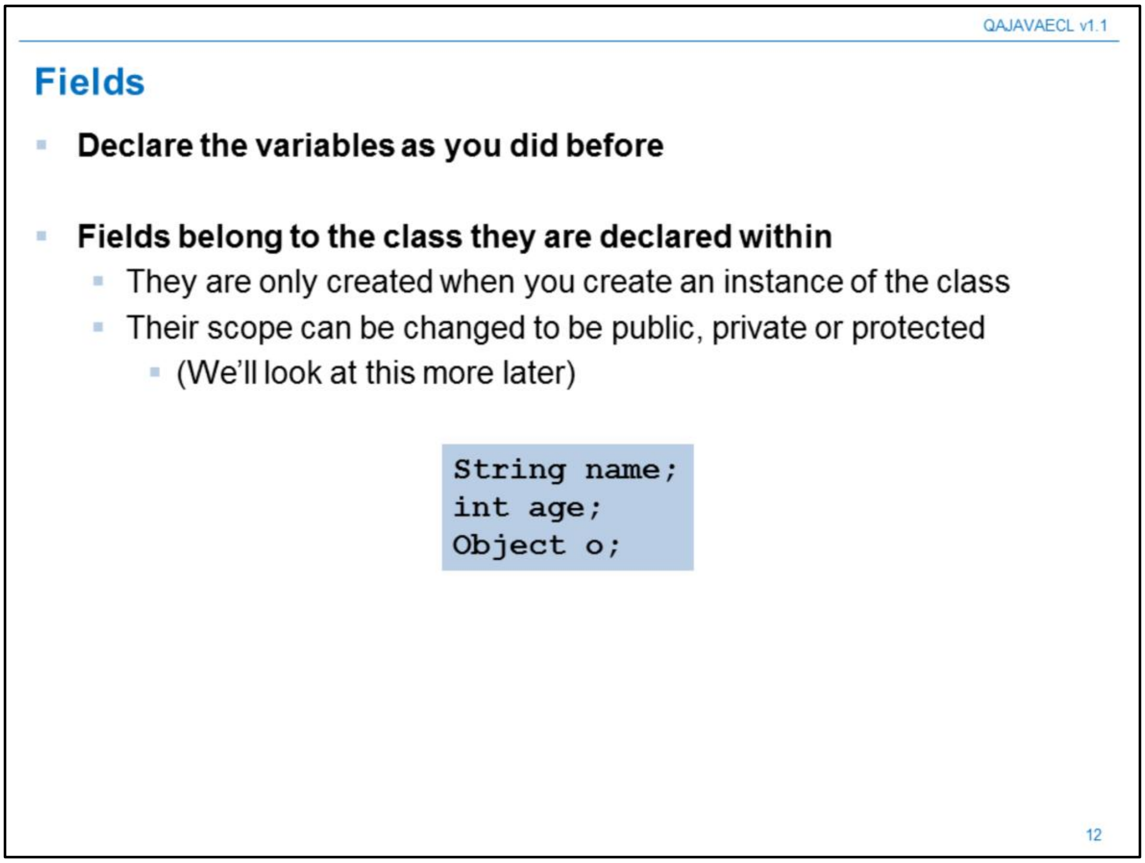


Java Class

- **A java class is a description of an object**
  - It contains the fields and methods
  - The class name should match the filename
    - Naming convention is to capitalise the words in a name

```
public class Rabbit {
        String name;
        int age;
        String colour;

        public Rabbit(String n){
                name = n;
        }

        public void increaseAge(){
                age = age + 1;
        }

        public int getAge() { return age; }
}
```

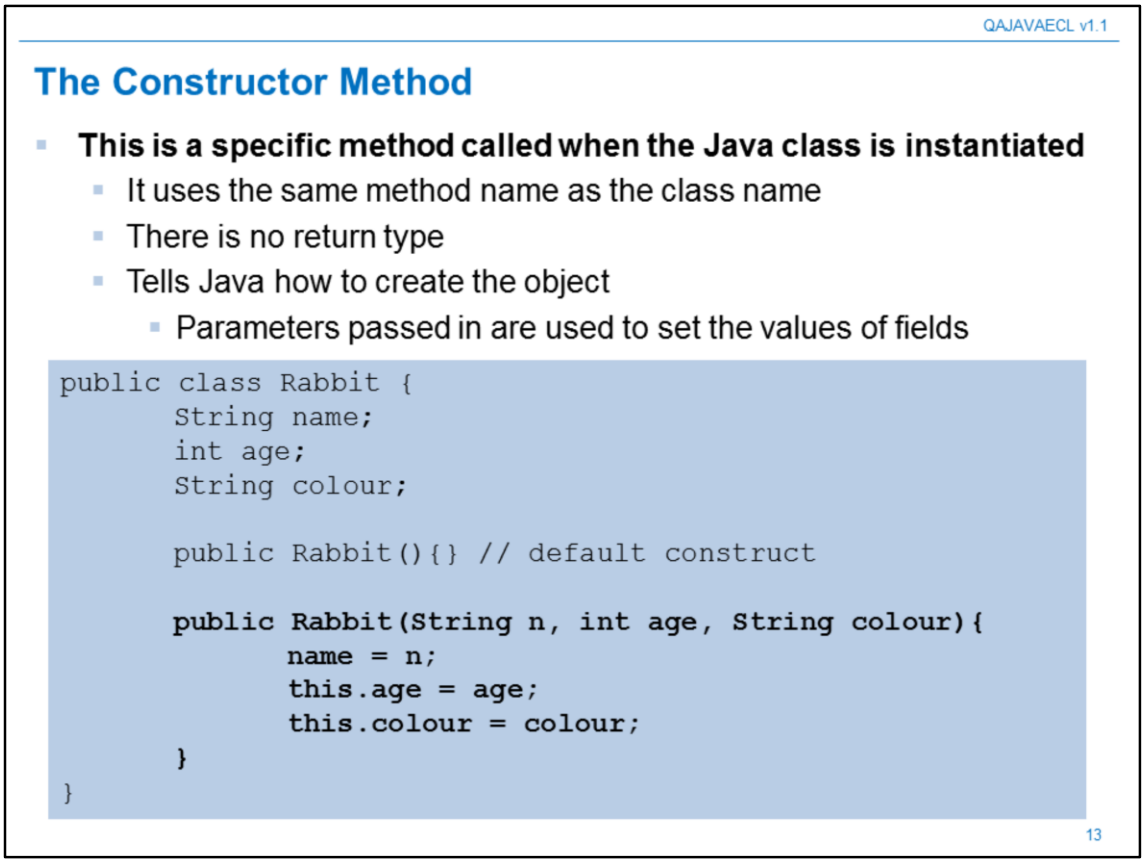
Fields
Constructor
Methods
Body



Fields

- **Declare the variables as you did before**
- **Fields belong to the class they are declared within**
  - They are only created when you create an instance of the class
  - Their scope can be changed to be public, private or protected
    - (We'll look at this more later)

```
String name;
int age;
Object o;
```



QAJAVAECL v1.1

The Constructor Method

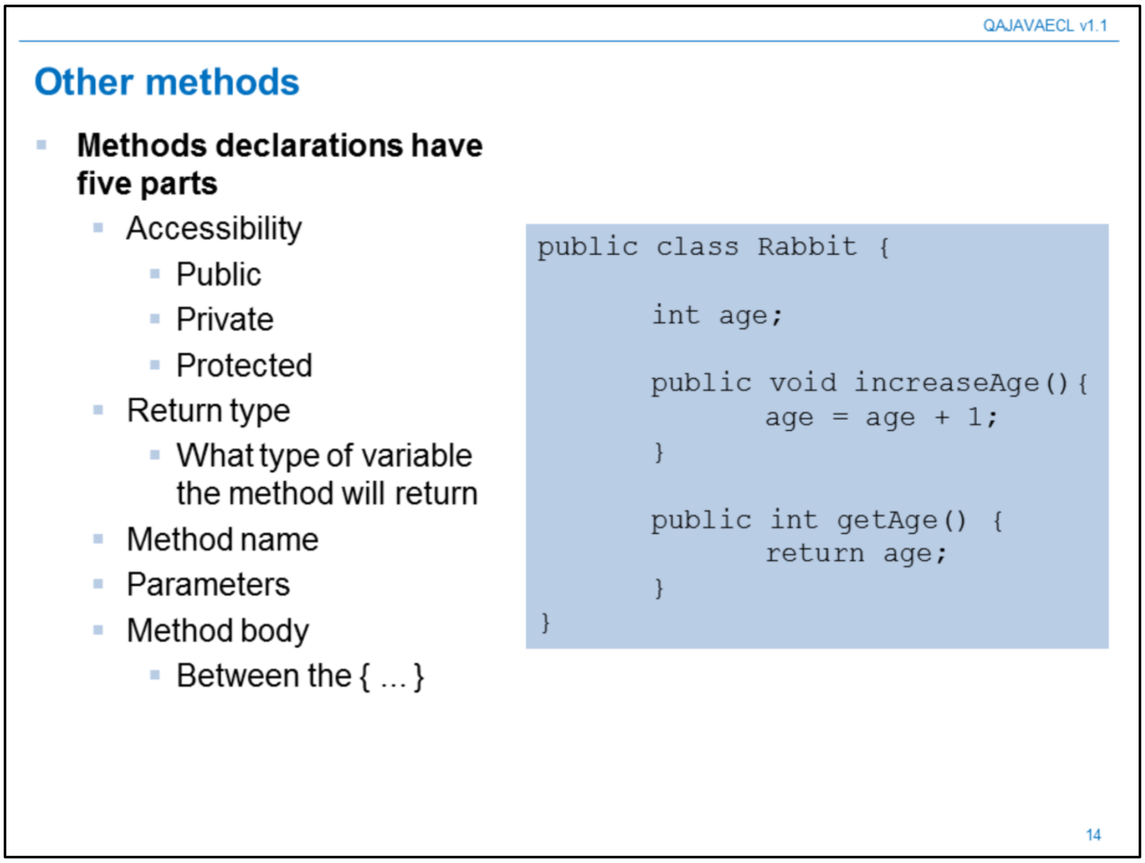
- **This is a specific method called when the Java class is instantiated**
  - It uses the same method name as the class name
  - There is no return type
  - Tells Java how to create the object
    - Parameters passed in are used to set the values of fields

```
public class Rabbit {
        String name;
        int age;
        String colour;

        public Rabbit(){} // default construct

        public Rabbit(String n, int age, String colour){
                name = n;
                this.age = age;
                this.colour = colour;
        }
}
```

13

Use the keyword “this” to refer to fields and methods specifically within this object. If we didn’t use “this.age” in the example above it would use age from the parameter on both the left hand side and the right hand side of the expression (achieving nothing!).

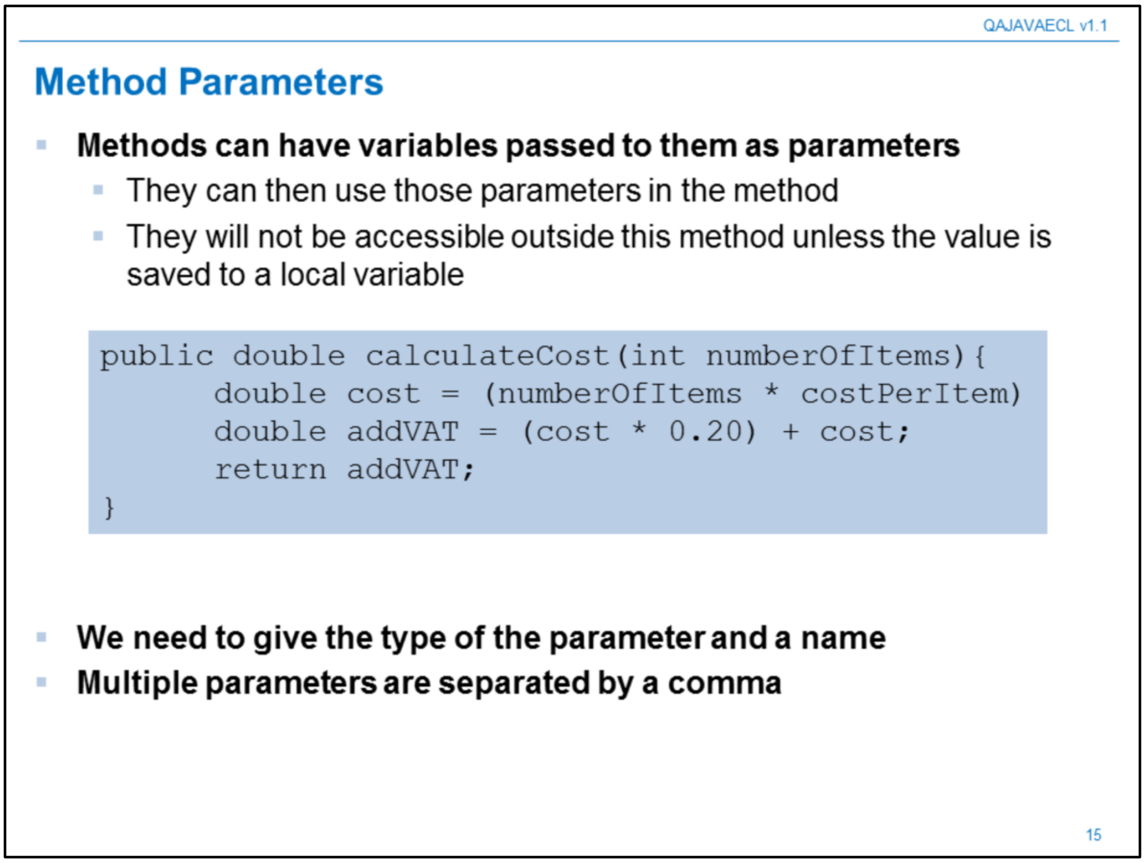
Variables are always accessed using the closest scoping first (parameters and variables declared in the method before those declared in the class)



Other methods

- **Methods declarations have five parts**
  - Accessibility
    - Public
    - Private
    - Protected
  - Return type
    - What type of variable the method will return
  - Method name
  - Parameters
  - Method body
    - Between the { … }

```
public class Rabbit {

        int age;

        public void increaseAge(){
                age = age + 1;
        }

        public int getAge() {
                return age;
        }
}
```



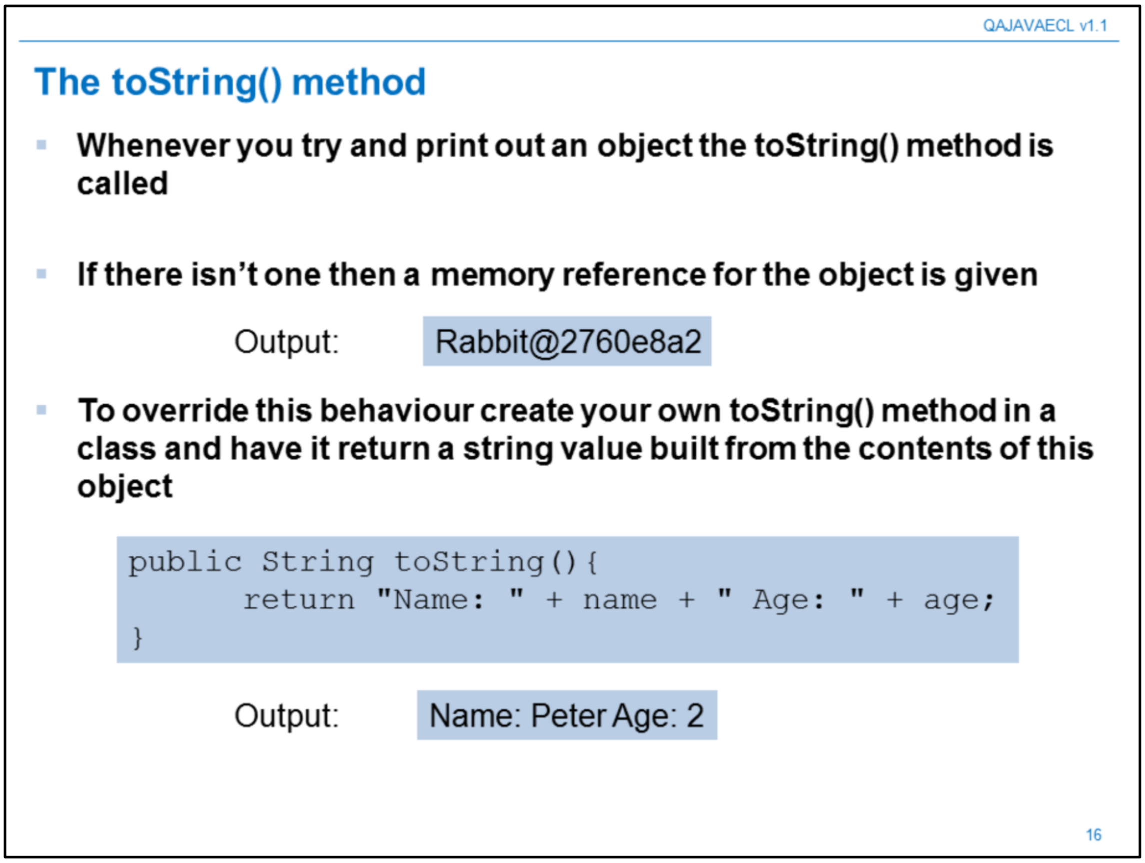
Method Parameters

- **Methods can have variables passed to them as parameters**
  - They can then use those parameters in the method
  - They will not be accessible outside this method unless the value is saved to a local variable

```
public double calculateCost(int numberOfItems){
      double cost = (numberOfItems * costPerItem)
      double addVAT = (cost * 0.20) + cost;
      return addVAT;
}
```

- **We need to give the type of the parameter and a name**
- **Multiple parameters are separated by a comma**

15



QAJAVAECL v1.1

The toString() method

- **Whenever you try and print out an object the toString() method is called**
- **If there isn't one then a memory reference for the object is given**

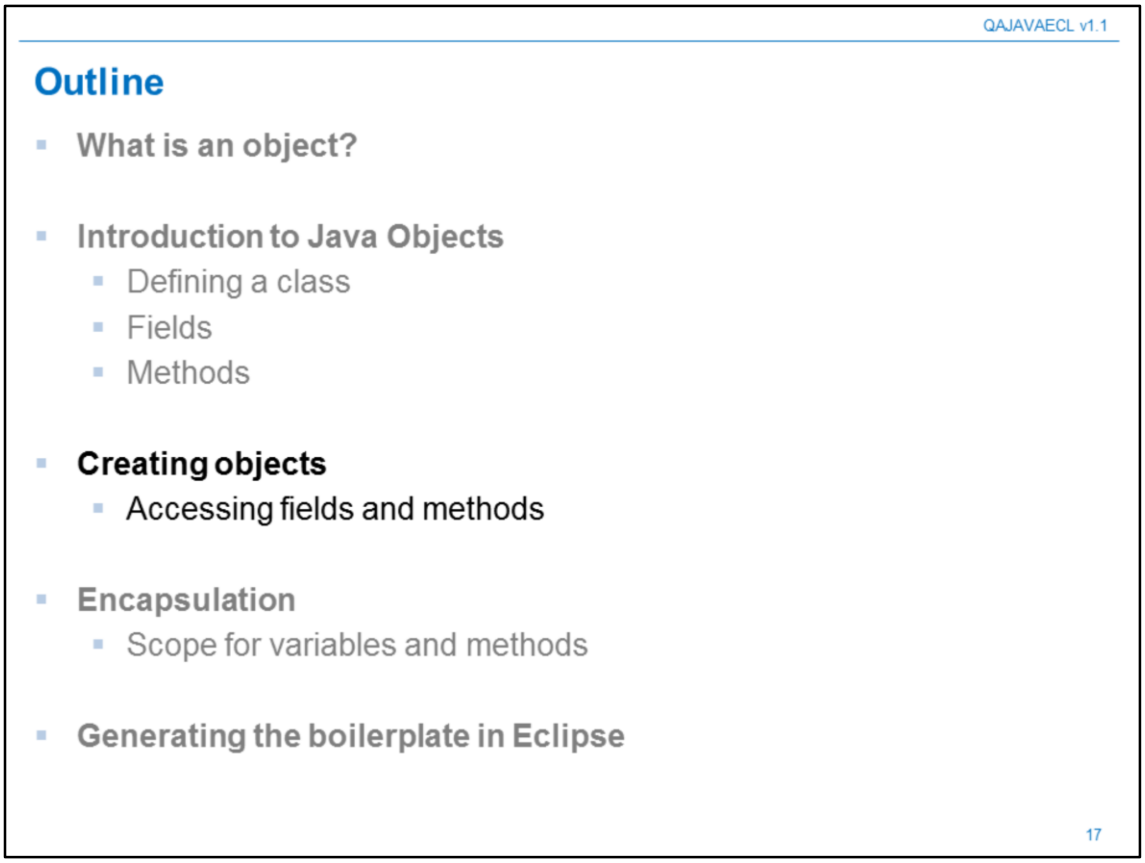
Output:
```
Rabbit@2760e8a2
```

- **To override this behaviour create your own toString() method in a class and have it return a string value built from the contents of this object**

```
public String toString(){
        return "Name: " + name + " Age: " + age;
}
```

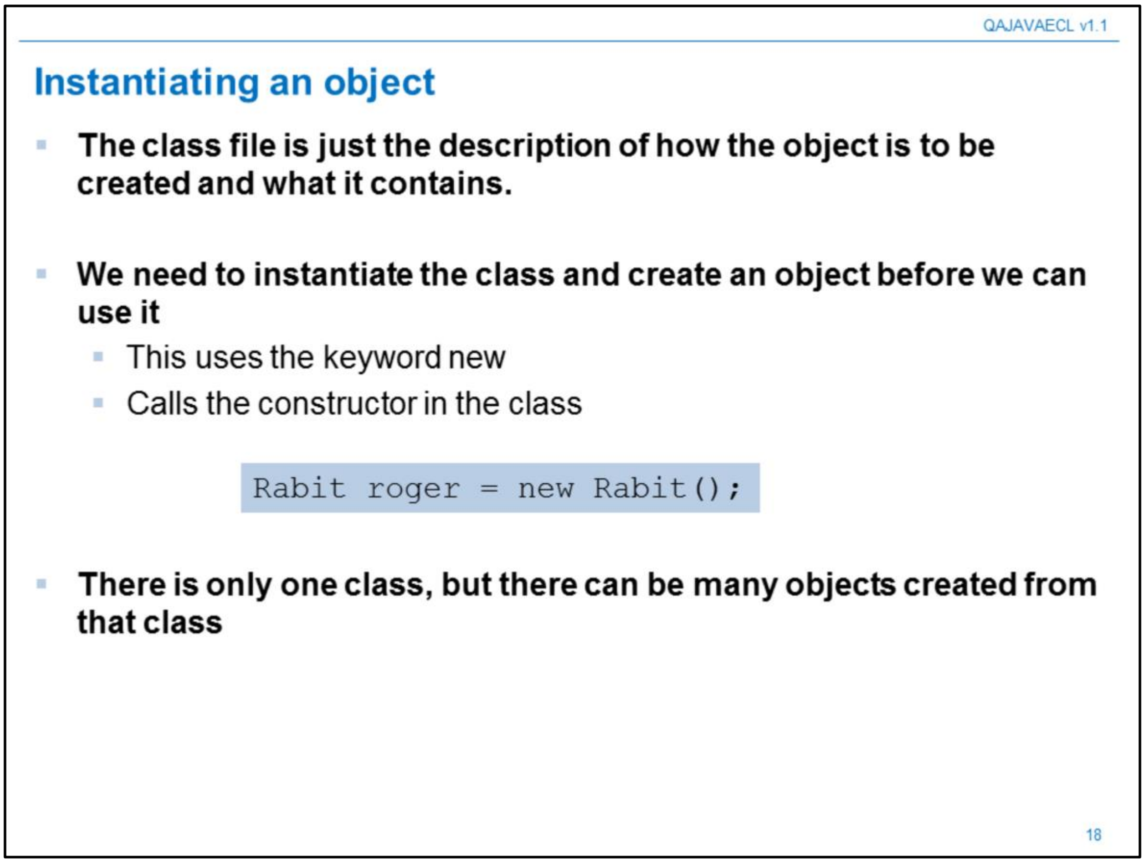
Output:
```
Name: Peter Age: 2
```

16



Outline

- What is an object?
- Introduction to Java Objects
  - Defining a class
  - Fields
  - Methods
- Creating objects
  - Accessing fields and methods
- Encapsulation
  - Scope for variables and methods
- Generating the boilerplate in Eclipse

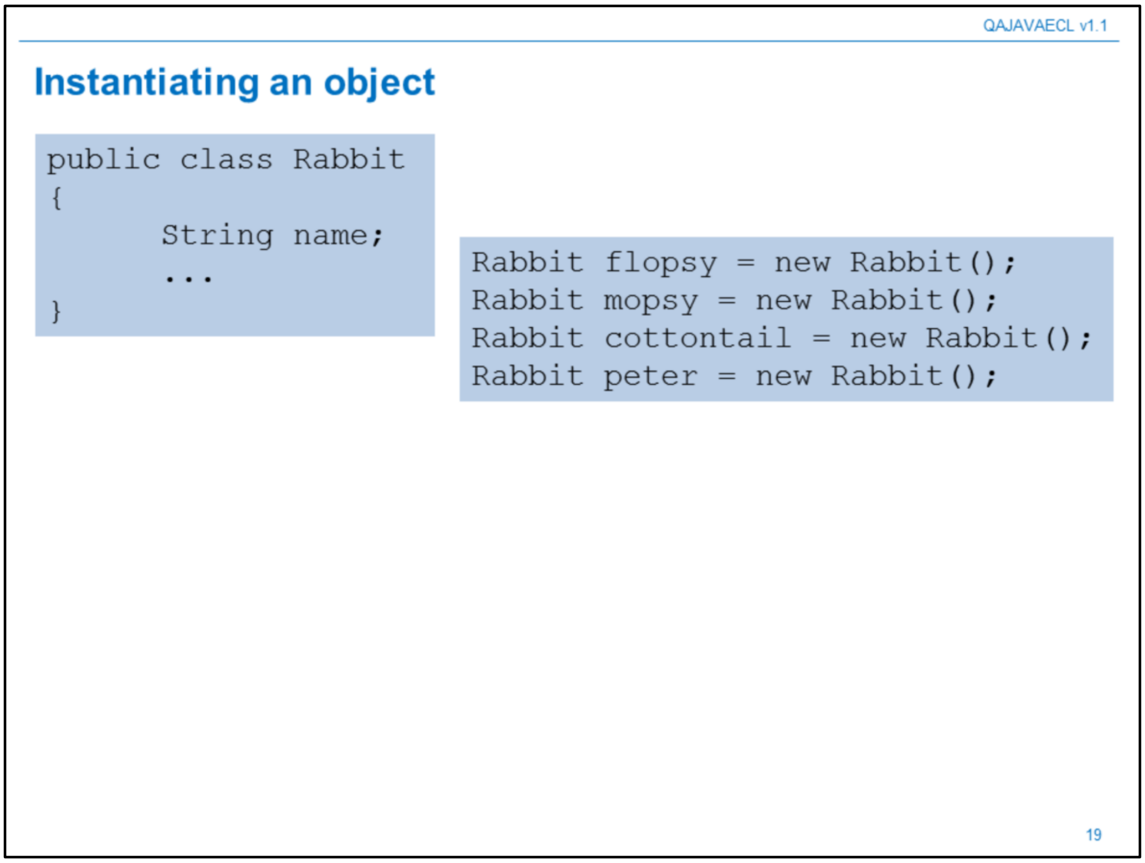


Instantiating an object

- **The class file is just the description of how the object is to be created and what it contains.**
- **We need to instantiate the class and create an object before we can use it**
  - This uses the keyword new
  - Calls the constructor in the class

```
Rabit roger = new Rabit();
```

- **There is only one class, but there can be many objects created from that class**



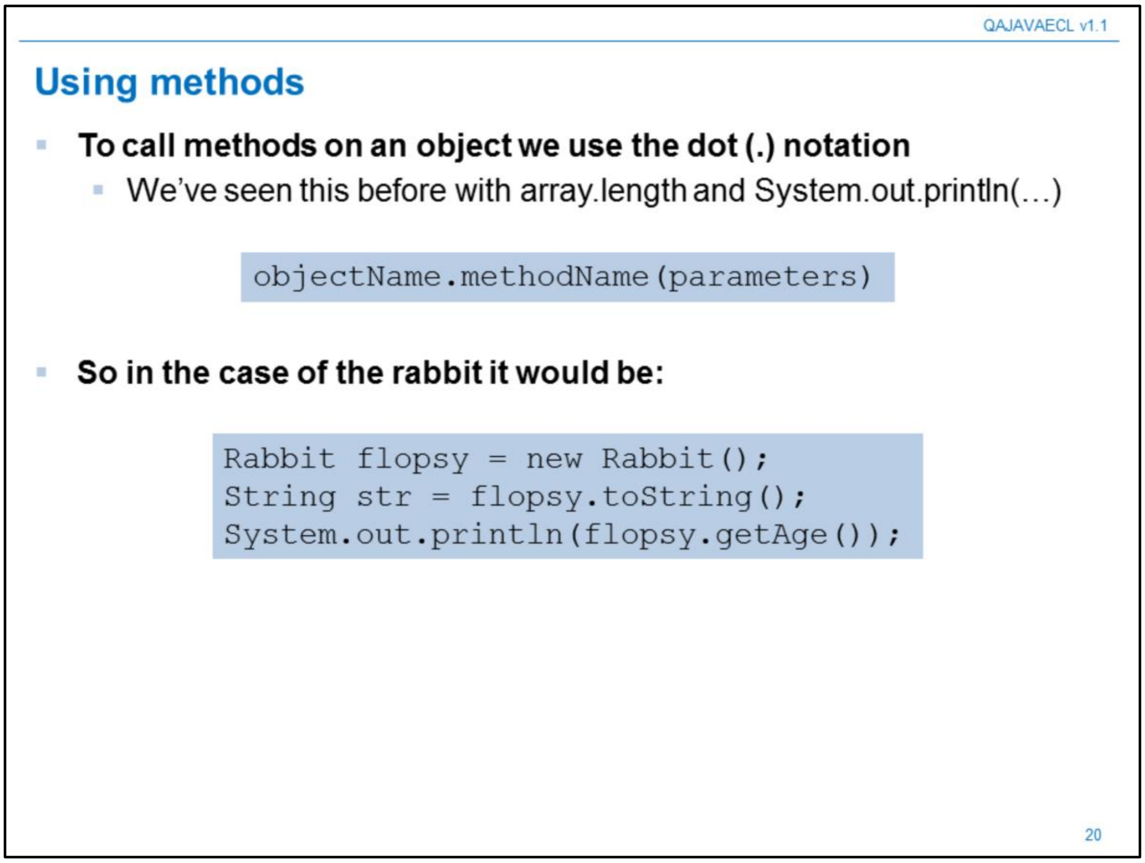
QAJAVAECL v1.1

Instantiating an object

```
public class Rabbit
{
        String name;
        ...
}
```

```
Rabbit flopsy = new Rabbit();
Rabbit mopsy = new Rabbit();
Rabbit cottontail = new Rabbit();
Rabbit peter = new Rabbit();
```

19



QAJAVAECL v1.1

Using methods

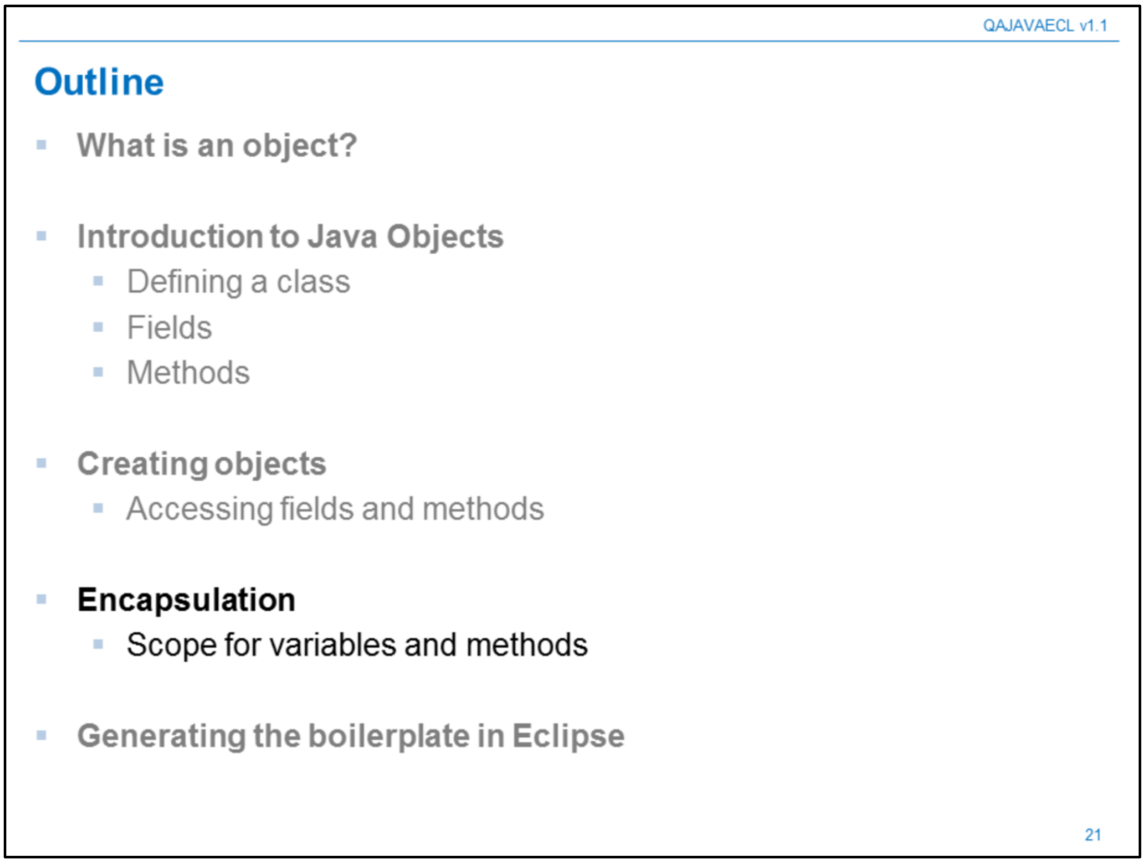
- **To call methods on an object we use the dot (.) notation**
  - We've seen this before with array.length and System.out.println(…)

```
objectName.methodName(parameters)
```

- **So in the case of the rabbit it would be:**

```
Rabbit flopsy = new Rabbit();
String str = flopsy.toString();
System.out.println(flopsy.getAge());
```

20

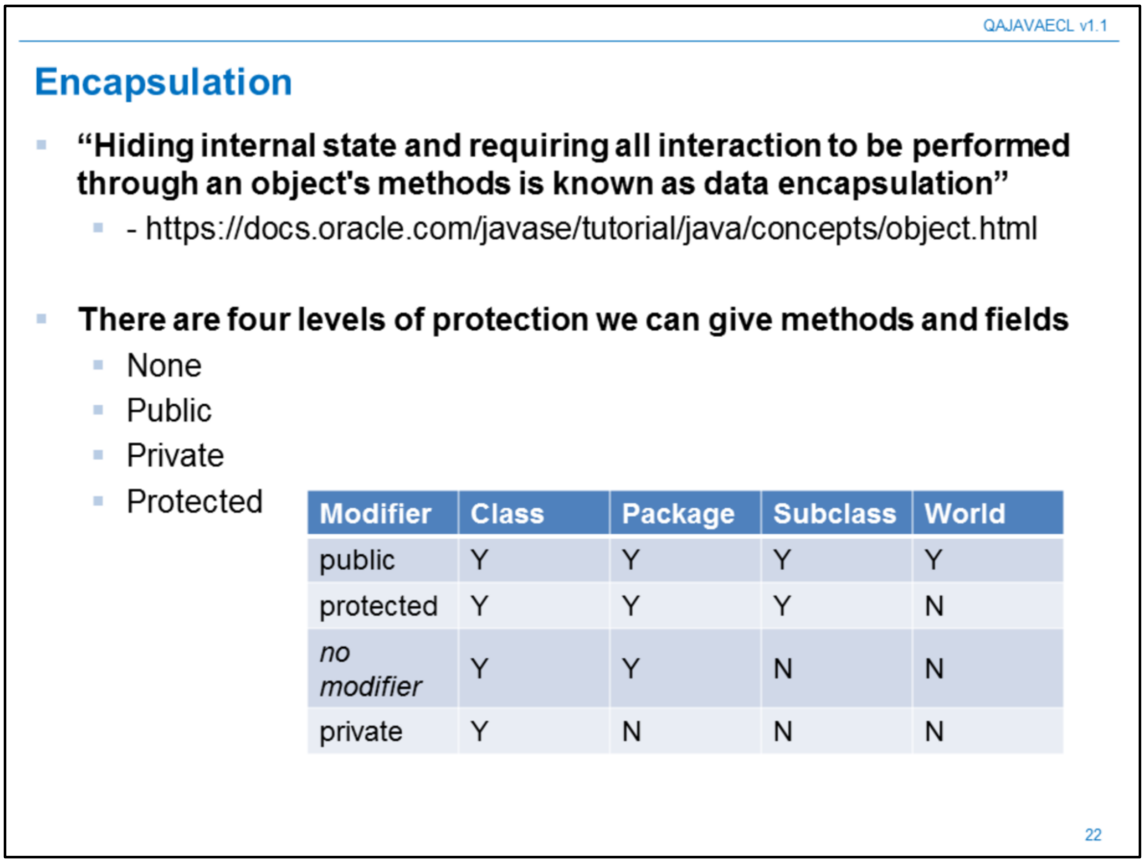


QAJAVAECL v1.1

Outline

- What is an object?
- Introduction to Java Objects
  - Defining a class
  - Fields
  - Methods
- Creating objects
  - Accessing fields and methods
- **Encapsulation**
  - Scope for variables and methods
- Generating the boilerplate in Eclipse

21



QAJAVAECL v1.1

Encapsulation

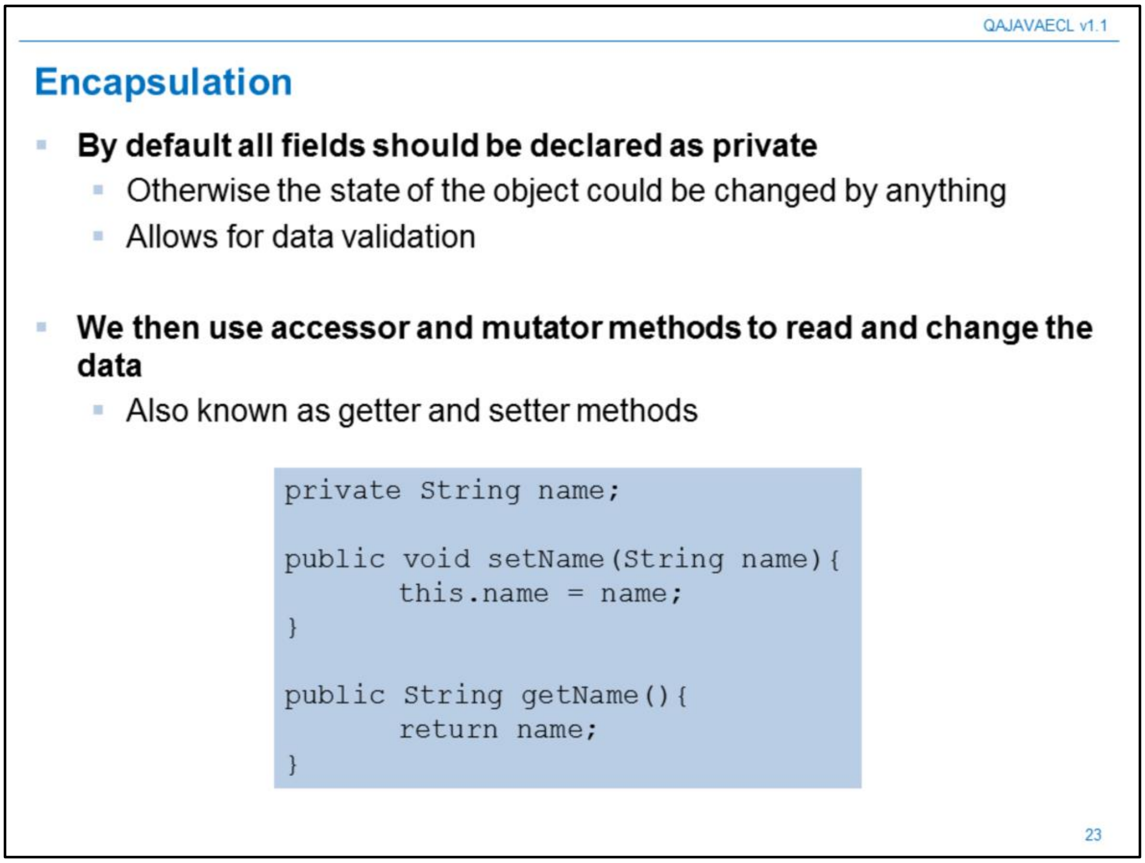
- **“Hiding internal state and requiring all interaction to be performed through an object's methods is known as data encapsulation”**
  - - https://docs.oracle.com/javase/tutorial/java/concepts/object.html

- **There are four levels of protection we can give methods and fields**
  - None
  - Public
  - Private
  - Protected

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

22

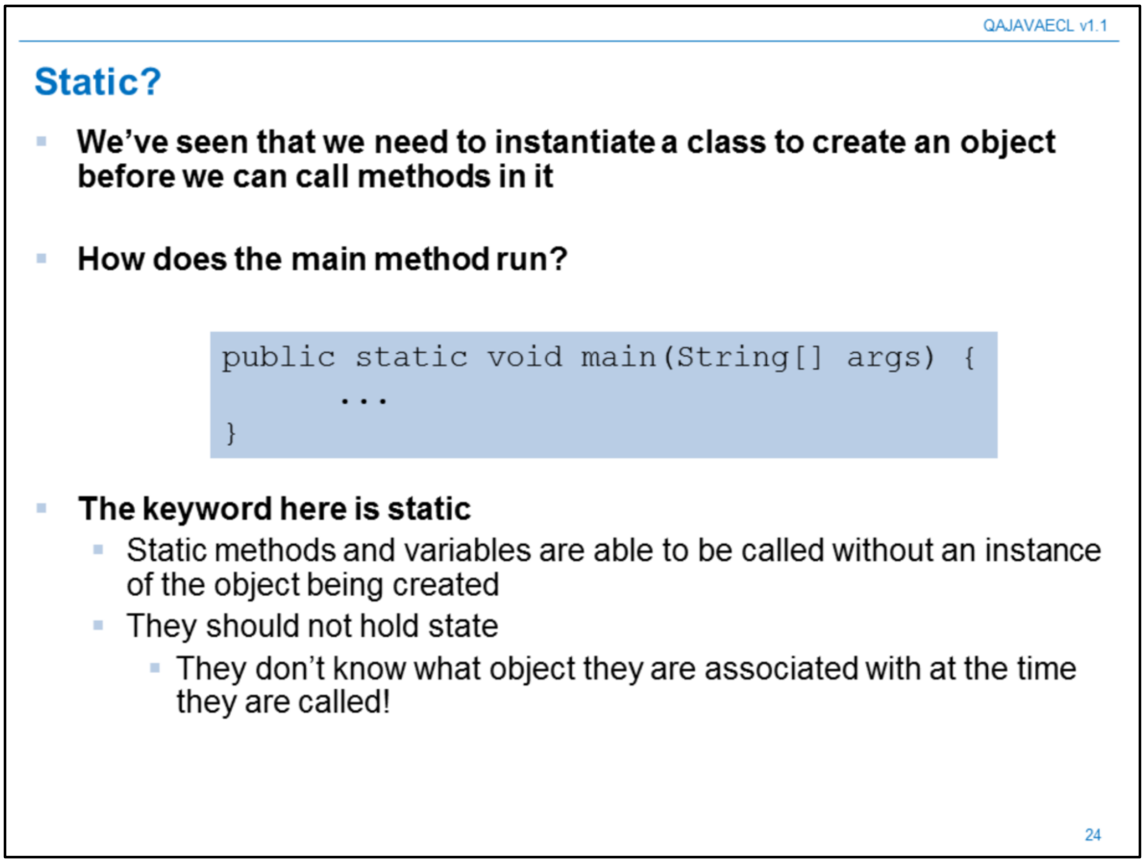
Table adapted from:
http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html



Encapsulation

- **By default all fields should be declared as private**
  - Otherwise the state of the object could be changed by anything
  - Allows for data validation
- **We then use accessor and mutator methods to read and change the data**
  - Also known as getter and setter methods

```
private String name;

public void setName(String name){
       this.name = name;
}

public String getName(){
       return name;
}
```



QAJAVAECL v1.1

Static?

- **We've seen that we need to instantiate a class to create an object before we can call methods in it**
- **How does the main method run?**

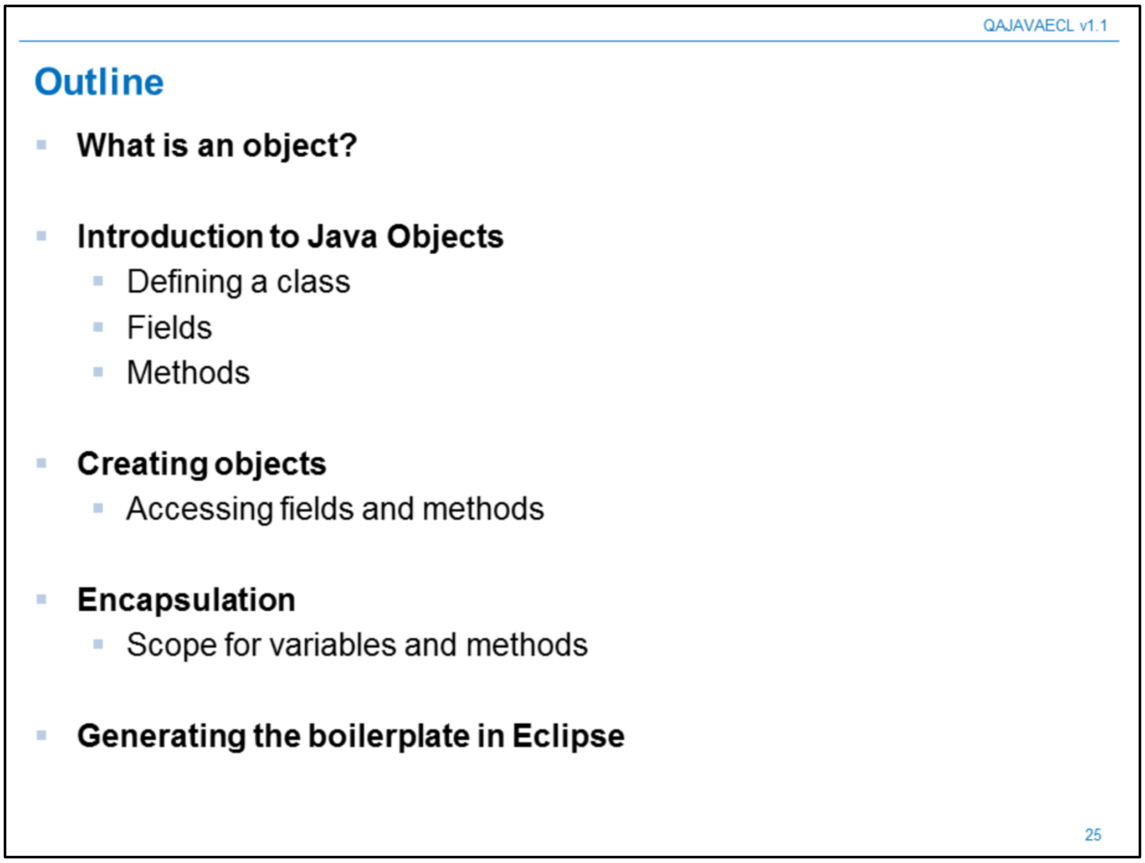
```
public static void main(String[] args) {
        ...
}
```

- **The keyword here is static**
  - Static methods and variables are able to be called without an instance of the object being created
  - They should not hold state
    - They don't know what object they are associated with at the time they are called!

24

Think of the static keyword as if the JVM is creating a brand new object in memory with just that method or field in it. When the method is completed, the box is thrown away rather than retained. This means that if you try and store something in a static context, the next time you call it the JVM will create another, brand new, object for you to use. The old information is not available anymore.

Static should be used with care. It is generally useful where we have 'helper methods' such as those that convert between one thing and another without needing to know the context it is working within. A method that converts an object from dollars to pounds doesn't need to know what we're using those numbers for, just that it gets something in and returns something. The previous calculated values will not be stored.



QAJAVAECL v1.1

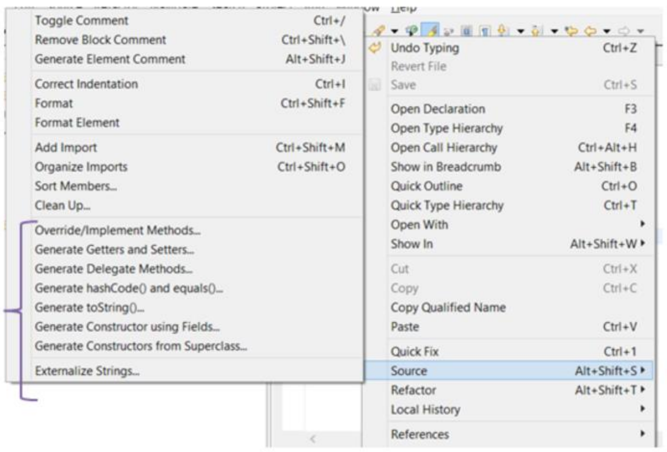
Outline

- **What is an object?**
- **Introduction to Java Objects**
  - Defining a class
  - Fields
  - Methods
- **Creating objects**
  - Accessing fields and methods
- **Encapsulation**
  - Scope for variables and methods
- **Generating the boilerplate in Eclipse**

25

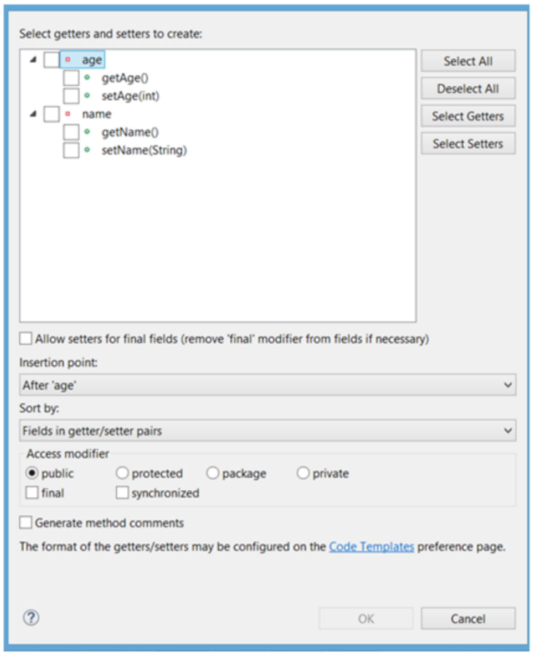
Using eclipse to generate the boilerplate

- **Boilerplate code refers to simple methods classes and annotation that you will end up writing over and over again**
- **Eclipse can automatically generate these for you!**
  - Be careful though, it may expose something you don't expect
  - Right click
    - Source Menu
  - Alt+Shift+S


Toggle Comment Ctrl+/
Remove Block Comment Ctrl+Shift+\
Generate Element Comment Alt+Shift+J
Correct Indentation Ctrl+I
Format Ctrl+Shift+F
Format Element
Add Import Ctrl+Shift+M
Organize Imports Ctrl+Shift+O
Sort Members...
Clean Up...
Override/Implement Methods...
Generate Getters and Setters...
Generate Delegate Methods...
Generate hashCode() and equals()...
Generate toString()...
Generate Constructor using Fields...
Generate Constructors from Superclass...
Externalize Strings...
Undo Typing Ctrl+Z
Revert File
Save Ctrl+S
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Show in Breadcrumb Alt+Shift+B
Quick Outline Ctrl+O
Quick Type Hierarchy Ctrl+T
Open With
Show In Alt+Shift+W
Cut Ctrl+X
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Quick Fix Ctrl+1
Source Alt+Shift+S
Refactor Alt+Shift+T
Local History
References

Generating the getter and setters

- Automatically generate all the get and set methods for each fields you've defined in the class
- We can even generate method comments
- The code templates can be configured to give different formats for the code



Select getters and setters to create:

- [ ] age
  - [ ] getAge()
  - [ ] setAge(int)
- [ ] name
  - [ ] getName()
  - [ ] setName(String)

[ ] Allow setters for final fields (remove 'final' modifier from fields if necessary)

Insertion point: After 'age'

Sort by: Fields in getter/setter pairs

Access modifier: [✓ public] protected package private

[ ] final [ ] synchronized

[ ] Generate method comments

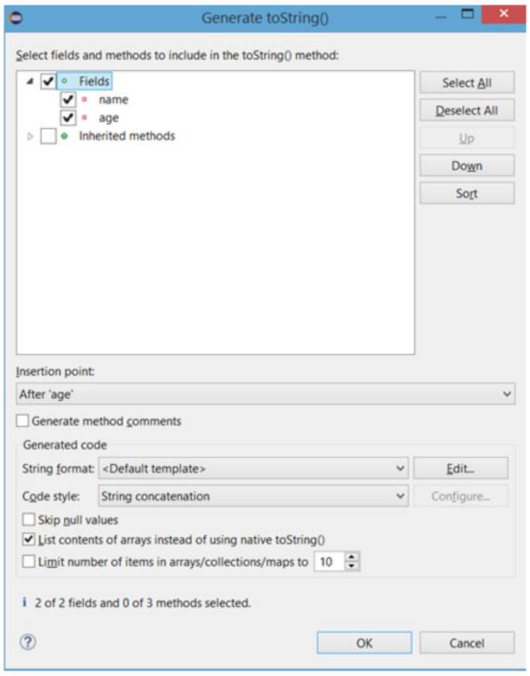
The format of the getters/setters may be configured on the Code Templates preference page.

Generating the toString() method

- **Generates a toString() method in the format:**

```
Classname [field=value,
                  field=value]
```

- **Select which parts of the class you wish to display**



Insertion point: After 'age'

[ ] Generate method comments

Generated code

String format: <Default template>

Code style: String concatenation

[ ] Skip null values

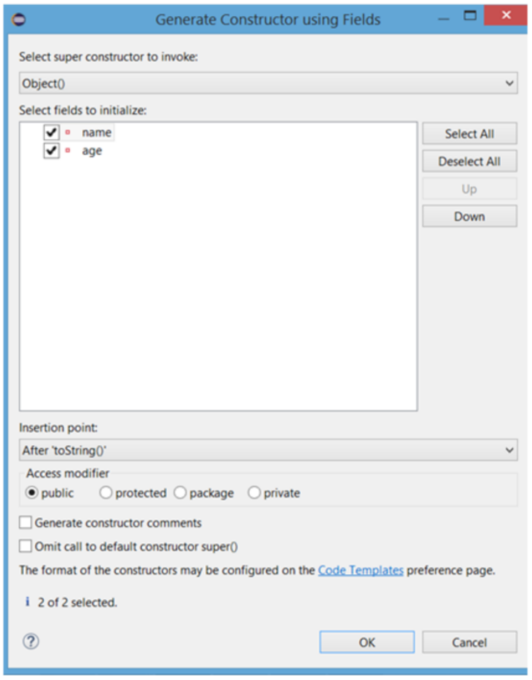
[✓] List contents of arrays instead of using native toString()

[ ] Limit number of items in arrays/collections/maps to 10

2 of 2 fields and 0 of 3 methods selected.

Generating the constructor

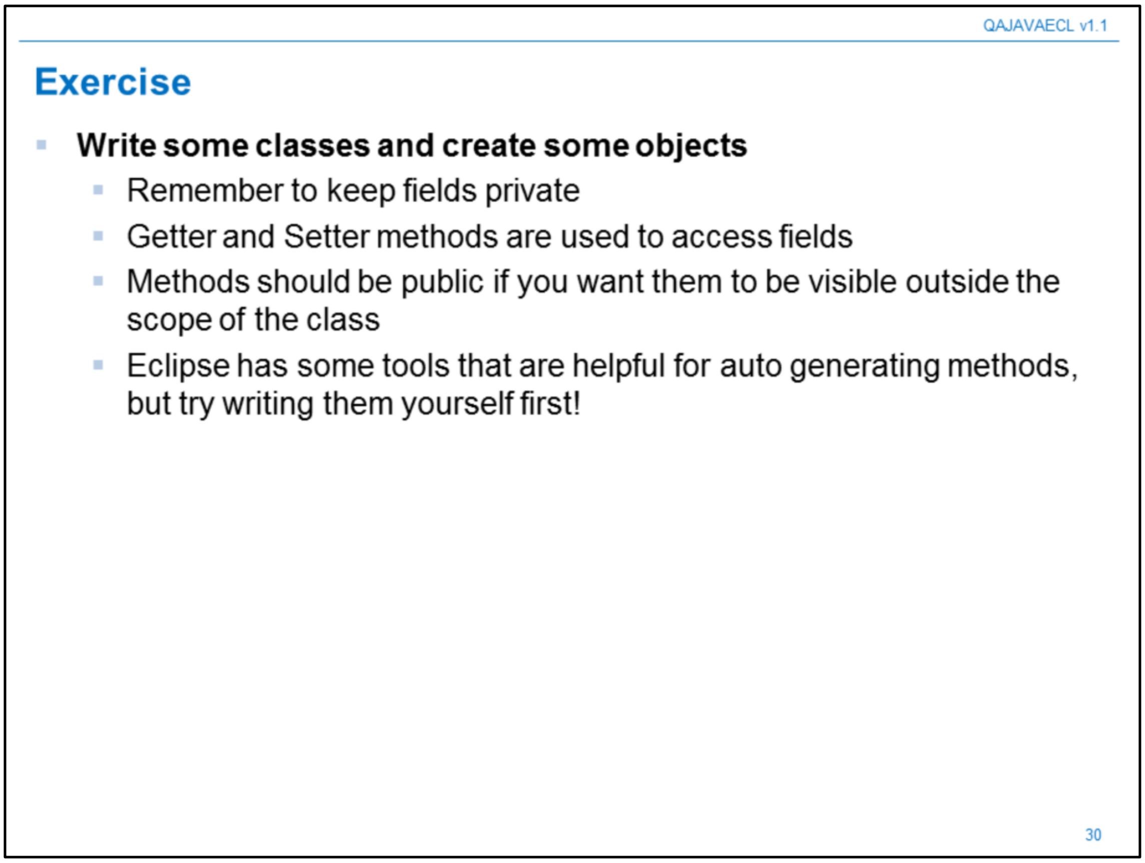
- **Can generate the constructor using the fields in the current class**
  - Will have a parameter for each field
  - Will set the value of the local variable
- **Alternatively, generates a constructor using the parent class**
  - This is part of the inheritance stuff we will go over later in the course


Generate Constructor using Fields
Select super constructor to invoke:
Object()
Select fields to initialize:
✓ name
✓ age
Select All
Deselect All
Up
Down
Insertion point:
After 'toString()'
Access modifier
● public ○ protected ○ package ○ private
☐ Generate constructor comments
☐ Omit call to default constructor super()
OK
Cancel

The format of the constructors may be configured on the Code Templates preference page.

2 of 2 selected.

29

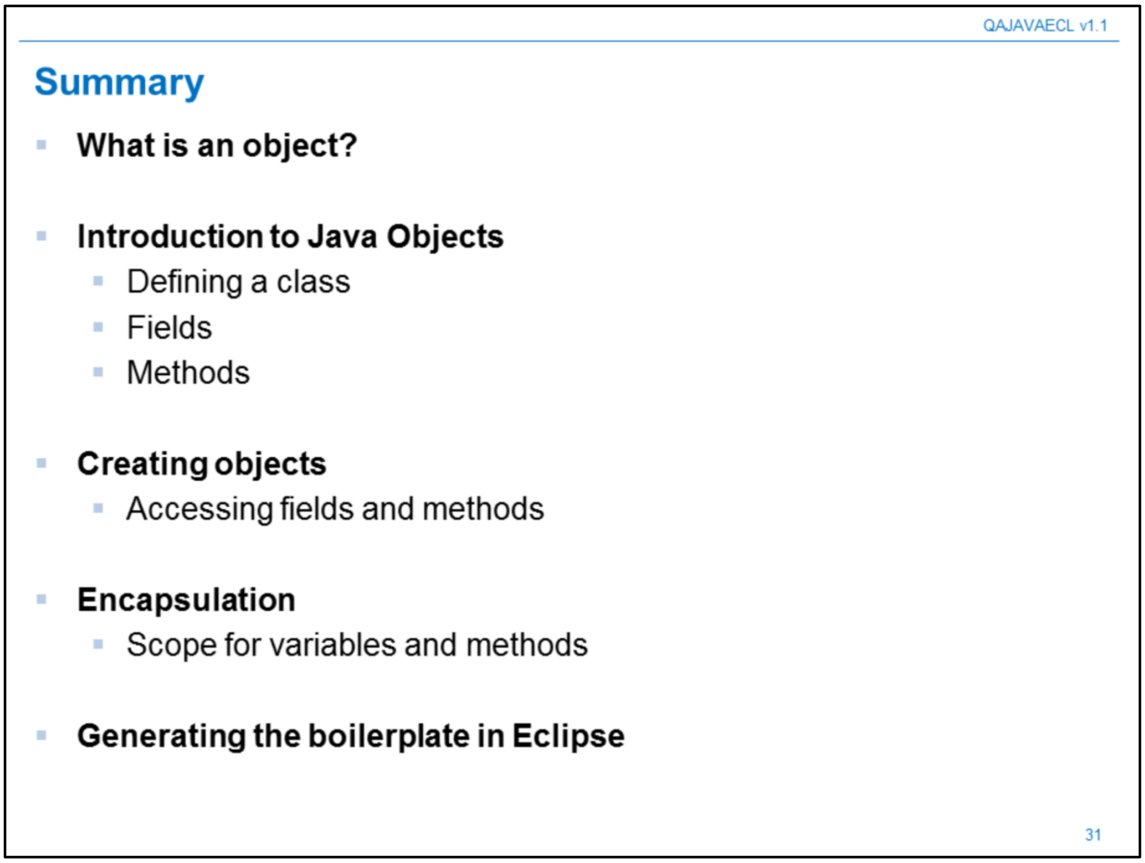


QAJAVAECL v1.1

Exercise

- **Write some classes and create some objects**
  - Remember to keep fields private
  - Getter and Setter methods are used to access fields
  - Methods should be public if you want them to be visible outside the scope of the class
  - Eclipse has some tools that are helpful for auto generating methods, but try writing them yourself first!

30



QAJAVAECL v1.1

Summary

- **What is an object?**
- **Introduction to Java Objects**
  - Defining a class
  - Fields
  - Methods
- **Creating objects**
  - Accessing fields and methods
- **Encapsulation**
  - Scope for variables and methods
- **Generating the boilerplate in Eclipse**

31