Object Oriented Programming

Objects, Classes and Methods

Dr. Seán Russell
sean.russell@ucd.ie

School of Computer Science,
University College Dublin

September XX, 2019

Table of Contents

1. Objects-Oriented Programming
2. Objects and Classes
   - Differences between Objects and Classes
3. Classes
   - Designing Classes
   - Adding Functionality to Classes
   - Constructors
4. Objects
   - Accessing Instance Variables
   - Calling Methods
5. Object References
6. Packages
7. Application Programmer Interface

Learning outcomes

After this lecture and the practical students should...

- understand the syntax of a Java class
- be able to define a basic class in Java
- be able to define constructors for a class in Java
- be able to define methods for a class in Java
- be able to call constructors and methods of a class in Java
- understand the concept of the implicit parameter and its use through the keyword `this`
- be familiar with how object references are stored and copied in Java
- understand the structure of Java code in packages
- be able to find information in the Java API

Table of Contents

1. Objects-Oriented Programming
2. Objects and Classes
3. Classes
4. Objects
5. Object References
6. Packages
7. Application Programmer Interface

Object-Oriented Programming

Writing large programs is hard, smaller pieces are easier

- Programs can be broken down into classes
- Classes can be broken down into related methods
- Methods are smaller and easier

What is Object-Oriented Programming?

- A different way of thinking about programming
- How we view and interact with the world
- Steve Jobs explained it this way:

Objects

Objects are like people. They're living, breathing things that have **knowledge inside** them about how to do things and have **memory inside** them so they can remember things. And rather than interacting with them at a very low level, you interact with them at a very high level of abstraction, like we're doing right here.

What is Object-Oriented Programming

Example Continued

Here's an example: If I'm your laundry object, you can give me your dirty clothes and send me a message that says, "Can you get my clothes laundered, please." I happen to know where the best laundry place in San Francisco is. And I speak English, and I have dollars in my pockets. So I go out and hail a taxicab and tell the driver to take me to this place in San Francisco. I go get your clothes laundered, I jump back in the cab, I get back here. I give you your clean clothes and say, "Here are your clean clothes."

What is Object-Oriented Programming

Example Continued

You have no idea how I did that. You have no knowledge of the laundry place. Maybe you speak French, and you can't even hail a taxi. You can't pay for one, you don't have dollars in your pocket. Yet I knew how to do all of that. And you didn't have to know any of it. All that complexity was hidden inside of me, and we were able to interact at a very high level of abstraction. That's what objects are. They encapsulate complexity, and the interfaces to that complexity are high level.

Table of Contents

1 Objects-Oriented Programming

2 Objects and Classes
- Differences between Objects and Classes

3 Classes

4 Objects

5 Object References

6 Packages

7 Application Programmer Interface

Breakdown of a Program

- Programs are broken into classes
- Classes contain a set of **related** functionalities
  - A laundry class can clean clothes, accept money...
- Functionalities are **related** to the purpose of the class
- This idea is called **cohesion**

Breakdown of a Class

- Classes can be broken down into methods
- The laundry class could have methods `cleanClothes` and `acceptMoney` or `addWater`, `addSoap` and `heatWater`
- Break problem into methods, then each method will be smaller
- Connect the smaller pieces together

Classes and Objects

Classes are similar to structs in C

Classes

A class can contain **instance variables**
A class can contain **methods** (functions)

Objects

An **object** is one **instance** of a class.
We can create many objects from the same class
Objects can have different values for instance variables

Table of Contents

1 Objects-Oriented Programming

2 Objects and Classes
- Differences between Objects and Classes

3 Classes

4 Objects

5 Object References

6 Packages

7 Application Programmer Interface

Differences between Objects and Classes

- A class like an idea
  - A Chair is something that we can sit on, it has legs and a back
  - There are billions of chairs in the world, each chair is an **instance** of the idea 'chair'
- In this way objects are the real versions of the ideas that are classes

Table of Contents

1. Objects-Oriented Programming
2. Objects and Classes
3. Classes
   - Designing Classes
   - Adding Functionality to Classes
   - Constructors
4. Objects
5. Object References
6. Packages

Classes

Every object is based on a class

- "Hello, World!" is based on the **String** class
- So are the objects "Goodbye" and "Mississippi"

Differences between objects

- Every object based on the String class has the same **methods** and **instance variables**
- The difference between each object is the **values** that are stored inside the **instance variables**

Table of Contents

1. Objects-Oriented Programming
2. Objects and Classes
3. Classes
   - Designing Classes
   - Adding Functionality to Classes
   - Constructors
4. Objects
5. Object References
6. Packages

Date struct example

Structs can be used to group variables

```
1 struct Date{
2   int day;
3   int month;
4   int year;
5 };
```

Date class Example

Here the `Date` class contains three **instance variables**; `day`, `month` and `year`.

```
1 class Date {
2   int day;
3   int month;
4   int year;
5 }
```

Table of Contents

1. Objects-Oriented Programming
2. Objects and Classes
3. Classes
   - Designing Classes
   - Adding Functionality to Classes
   - Constructors
4. Objects
5. Object References
6. Packages

Methods

- We can define **methods inside** a class
- Not possible in C, must use functions and a struct
- Example: A function to increment the date, called `incrementDay`
- The function requires a parameter to know which struct variable to change
- `void incrementDay(struct Date* d)`

Data and Functionality Together

- Functionality and data together
- Do not need to pass a Date object as a parameter

```
1 class Date {
2   int day;
3   int month;
4   int year;
5   void incrementYear(){}
6   void incrementMonth(){}
7   void incrementDay(){}
8 }
```

Struct with function in C

```
1  struct Date{
2     int day;
3     int month;
4     int year;
5  };
6  void incrementMonth(struct Date* d){
7     d->day = 1;
8     if (d->month < 12) {
9        d->month++;
10    } else {
11       incYear(date);
12    }
13 }
```

Struct with function in C

- A struct has to be passed as a parameter
- First the day counter returns to 1
- Changed in parameter `d` using `d->day = 1;`.
- Then check if `month` < 12 (before December) and
  - increment month in `d` **or**
  - pass `d` to another function to increment the year

Class with method in Java

```
1  class Date {
2     int day;
3     int month;
4     int year;
5     ...
6     void incrementMonth() {
7        day = 1;
8        if (month < 12) {
9           month++;
10       } else {
11          incrementYear();
12       }
13    }
14 }
```

Class with method in Java

- There is **no parameter** for this method, How does the method know what the values of day, month and year are?
- Because the method and the instance variables are part of the same class, it will always change the values for the object that the method is called for
- This way we use `day` to access or change the value of day and so on

Adding methods to classes

Methods can only be defined **in a class**

- Specify the **return type**
- Specify the **name** of the method
- Specify the **parameters** required
- Then we add the code

Example

For example a method for incrementing the year of a date object

- What is the return type?
- What is the name of the method?
- What parameters does it need to work?

Increment year example

Example

For example a method for incrementing the year of a date object

- What is the return type? `void`
- What is the name of the method? `incrementYear`
- What parameters does it need to work? **none**

Method signature

```
void incrementYear(){}
```

Increment year example

After the method **signature** we can add the code

- What happens when the year changes?
  - The year increases by 1
  - The month goes back to 1
  - The day goes back to 1

Increment year example

```
1 void incrementYear() {
2   day = 1;
3   month = 1;
4   year++;
5 }
```

Table of Contents

1. Objects-Oriented Programming
2. Objects and Classes
3. Classes
   - Designing Classes
   - Adding Functionality to Classes
   - Constructors
4. Objects
5. Object References
6. Packages

Creating Objects

- To create an object, we need to use a **constructor**
- Constructors require the correct parameters
- Every class has a default constructor
- When we define a new constructor, the default constructor is removed
- We can add many constructors

Why add a Constructor?

- We define constructors because they can **require** information
- A Date object does not make sense without a value for day, month and year
- A constructor can force the users of the class to provide this information

Type of an Object

- When we declare a variable, we must declare its type (E.g. **int** x;)
- But what type do we use for objects?
- We use the name of the **class**
- E.g. if we create a date object, the type of the object is `Date`
- The type of an object determines the methods and instance variables available

Constructing objects

Creating a new object is called **constructing** the object

Constructing a Date object

To construct an object we use the keyword `new`

e.g. `new Date();`

We usually store the object in a variable

e.g. `Date today = new Date();`

This uses the default constructor and passes no information to the object

Values of Instance Variables

- What are the values of the instance variables?
- Instance variables are given a default value
- A date object with the value 00/00/00 is not useful
- This is why we add constructors
- It is reasonable that we **need** a value for day, month and year

Defining a Constructor

- Defining a constructor is like a method
- No return type
  - The object will be constructed, we do not **return** anything
- The **name** must match the class
  - How else would Java know it was a constructor
- We only need to decide what parameters are required

Defining a Constructor

- In this example, the class requires 3 ints
- Parameters should have different names
  - Instead of day, month and year, we can use d, m, and y

Unfinished Date Constructor

```
1 Date(int d, int m, int y) {
2
3 }
```

- The constructor must be defined **inside** the class
  - Between the matched { and }

Using the Constructor

- Now we can't construct a Date object unless we pass it three ints
- E.g. `Date b = new Date(21, 7, 1985);`
- The positions of the values relate to the parameters, d $\leftarrow$ 21, m $\leftarrow$ 7 and so on

Saving Parameters

- We must assign parameters to instance variables, E.g. `day = d;`

Saving parameters

```
1 Date(int d, int m, int y) {
2   day = d;
3   month = m;
4   year = y;
5 }
```

Table of Contents

1. Objects-Oriented Programming
2. Objects and Classes
3. Classes
4. Objects
   - Accessing Instance Variables
   - Calling Methods
5. Object References
6. Packages

Objects

- Objects are complex values that we can use
- We can call their methods or directly access the values of their instance variables
  - These are defined by the **type** (class) of the object

What can I do with an Object?

- The definition of a **class** decides what an **object** can do
- If a class defines 10 instance variables, then every object based on that class contains values for those variables
- If a class defines a method named `shout`, then we can use that method on any object based on that class

Table of Contents

1. Objects-Oriented Programming
2. Objects and Classes
3. Classes
4. Objects
   - Accessing Instance Variables
   - Calling Methods
5. Object References
6. Packages

Accessing Instance Variables

- The syntax for accessing a value inside an object is: `objectName.varName`
  - `objectName` is the name of the variable
  - `varName` is the name of the instance variable we want
- If we create a Date object named `today`,

```
Date today = new Date(23,9,2016);
```

- We can access `day` using the expression `today.day`
  - E.g. `System.out.println(today.day);`
  - E.g. `today.day = 21;`

Creating Objects

Default or Defined Constructor

Using default constructor

```
1 Date today = new Date();
2 today.day = 29;
3 today.month = 9;
4 today.year = 2017;
```

Using defined constructor

```
1 Date today = new Date(12, 9, 2016);
```

Table of Contents

1. Objects-Oriented Programming
2. Objects and Classes
3. Classes
4. Objects
   - Accessing Instance Variables
   - Calling Methods
5. Object References
6. Packages

Calling Methods

- In C if we had a function `max`, we would call it using `max(1,2);`
- In OOP, methods only exist as **part of a class**
- This means that to call a method, we must first have an object based on that class

Calling a method

- To call a method we need
  - A reference to the object (e.g. `Date today;`)
  - The name of the method
  - The parameters
- Once we have this information, we use the syntax: `objectName.methodName(parameters);`

Examples

```
1 Date d = new Date(21,9,2017);
2
3 d.incrementYear();
4 // This increases the year in d by 1
5 d.incrementMonth();
6 // This increases the month in d by 1
```

What values are used?

- An object is just one instance (copy) of a class
- Based on the same class we can create many different objects
- Every object has the same instance variables
- Each object can have different values for their instance variables

Two Date Objects

```
1 Date today = new Date(7, 4, 2016);
2 Date tomorrow = new Date(8, 4, 2016);
3
4 for(int i = 0; i < 100; i++){
5   today.incrementDay();
6 }
7
8 System.out.println(today.day + "/" +
    today.month + "/" + today.year);
9 System.out.println(tomorrow.day + "/" +
    tomorrow.month + "/" + tomorrow.year);
```

Table of Contents

1 Objects-Oriented Programming

2 Objects and Classes

3 Classes

4 Objects

5 Object References

6 Packages

7 Application Programmer Interface

Pointers in Java

- In Java we do not have to think about pointers
- This does not mean that there are no pointers
  - In Java pointers are called **references**
- In Java **every object is a reference**

What is a reference?

- When an object is constructed (using **new**), that object is created and stored in memory
- A constructor returns a **reference** to where the object is in memory
- Think of references as memory addresses

Copying a primitive variable

- Primitive variables are stored directly in memory

Copying a primitive variable

```
1 int x = 120;
2 int y = x;
3 y++;
```

- The **value** of 120 is stored in the variable `x`
- The **value** of `x` is then copied into `y`
- When `y` is incremented, only the value of `y` changes

Copying an object variable

Copying an object variable

```
1 Date t =  new Date(11, 9, 2018);
2 Date b = t;
3 b.incrementDay();
```

- The **address** of the object is stored in the variable `t`
- The **address** stored in `t` is then copied into `b`
- There is only one object in memory, but two variables that point to it
- When `b` is incremented, the value in the object is changed

Copying an object

- We cannot copy an object like a primitive variable
- We must create another object using the constructor

Copying an object

```
1 Date t = new Date(11,9,2018);
2
3 Date b = new Date(t.day, t.month, t.year);
```

Null references

- A reference has no value until we assign one
- A special value called `null` is used to mean no address
- Try to use a `null` variable and our program will **fail**
- This is called a `NullPointerException`

Table of Contents

1. Objects-Oriented Programming
2. Objects and Classes
3. Classes
4. Objects
5. Object References
6. Packages
7. Application Programmer Interface

Packages

- In Java we use **packages** to group code
- Each package is a folder that we put related code into
- This helps keep a lot of files sorted into the right places

Packages and folders

When we group files into packages we have to put the file into the correct folder.
e.g. If we have a package called `examples`, then any file in this package must be stored in a folder named `examples`

Packages

If we want a Java file to belong to a package, then we have to put a declaration at the start of the file.

- In this example the class `Test` is declared as being part of the `examples` package
- This means that the file Test.java must be stored in the folder examples

```
1 package examples;
2
3 class Test{
4
5 }
```

More Complicated Packages

If there are still too many files in a package we can separate the files into smaller packages.

- These packages are called **sub-packages**

- This means that we can put a file in a sub-package

More Complicated Packages

Example

For every lecture I prepare a lot of examples, if I put them all into the examples package there will be a lot of files. I can put my files from week 1 in a sub-package called week1

- This means my declaration would be `package examples.week1;`
- And the file would be stored in a folder week1 which is in a folder called examples

Packages in Java library

Java uses the same idea for storing all of its libraries

- The code that is contained inside each package is usually related

Examples

Package	Description
java.io	Java code that supports input and output
java.util	Java utilities, such as data structures and algorithms
java.text	Java code for processing and formatting text
java.lang	The core functionality of the Java language

Table of Contents

1 Objects-Oriented Programming

2 Objects and Classes

3 Classes

4 Objects

5 Object References

6 Packages

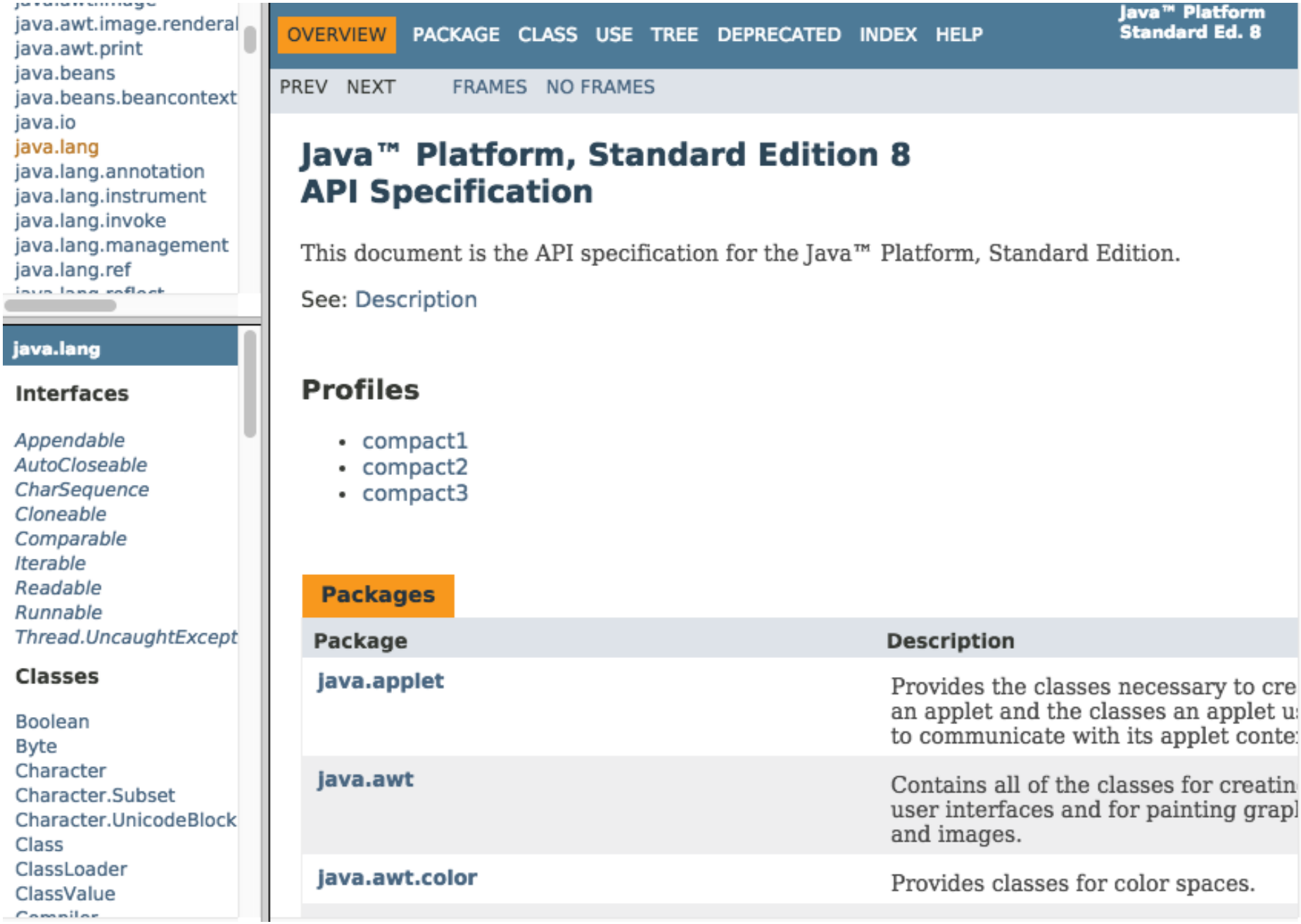
7 Application Programmer Interface

Application Programmer Interface (API)

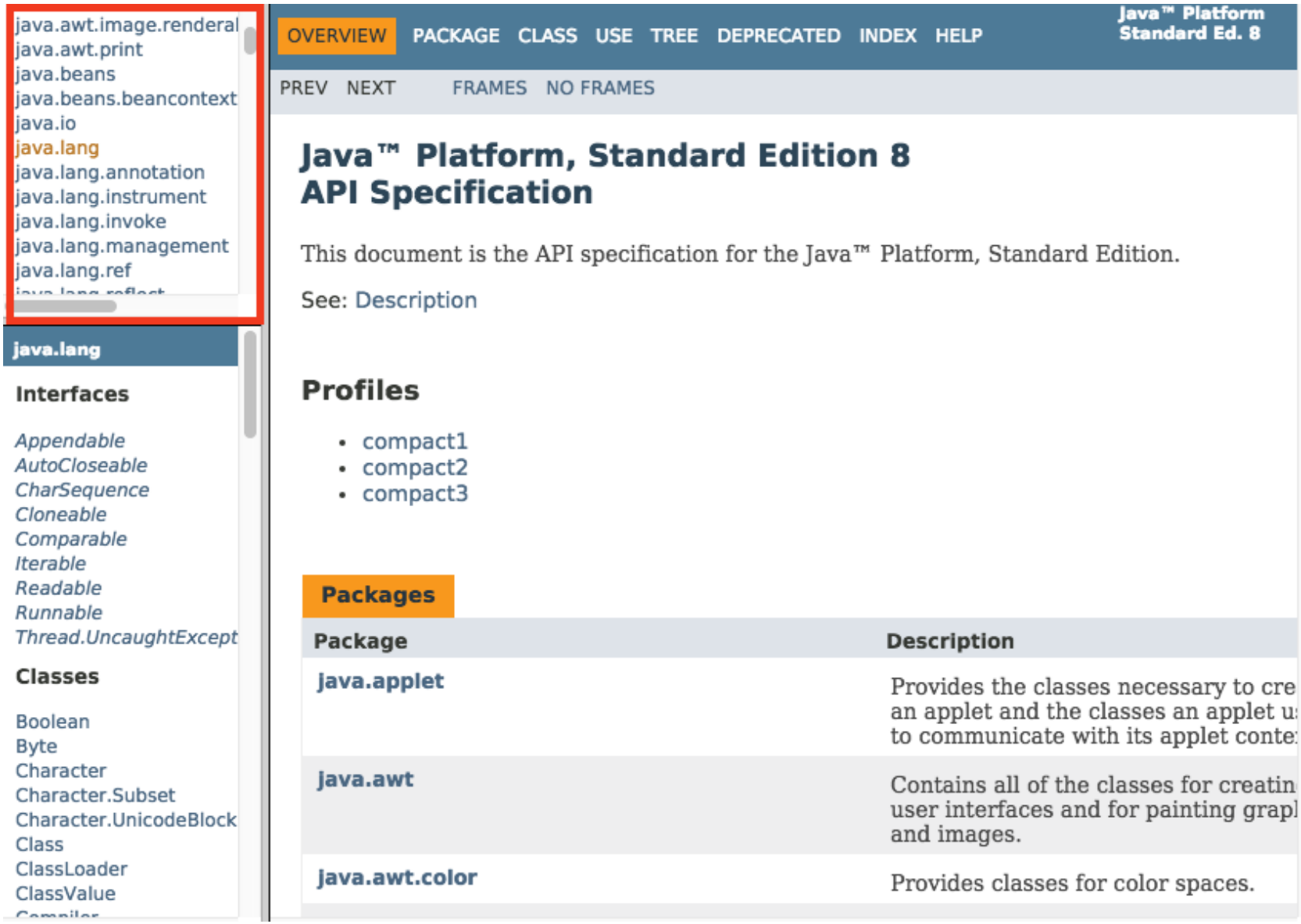
The application programmer interface (API) is the list of all of the libraries made available in Java

- The API lists all of the classes of the Java library
- For each class it also lists their `instance variables`, `constructors` and `methods`
- `https://docs.oracle.com/javase/8/docs/api/`

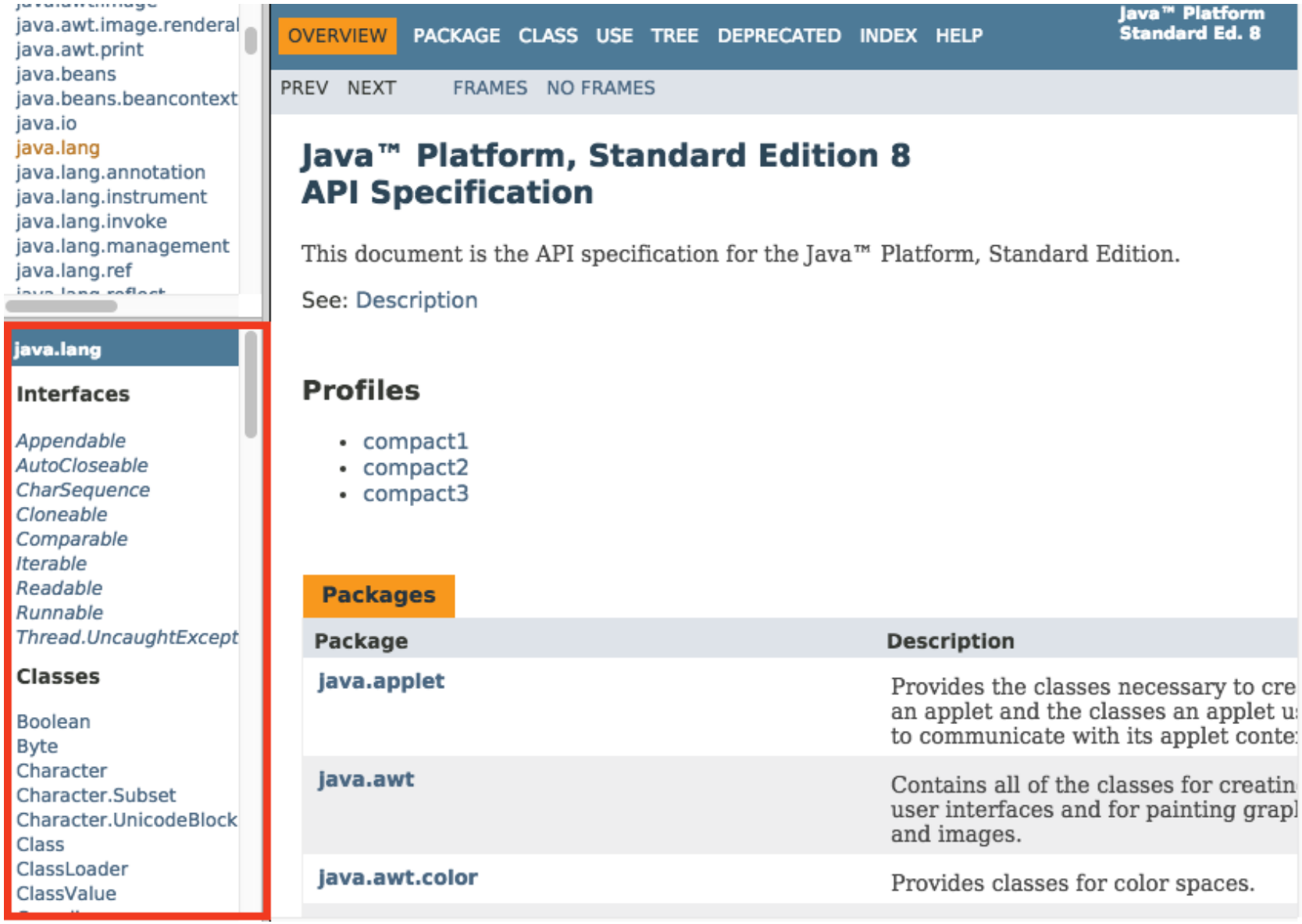
This documentation contains all of the information that we need to use any class in the library


java.awt.image.renderal
java.awt.print
java.beans
java.beans.beancontext
java.io
java.lang
java.lang.annotation
java.lang.instrument
java.lang.invoke
java.lang.management
java.lang.ref
java.lang
Interfaces
Appendable
AutoCloseable
CharSequence
Cloneable
Comparable
Iterable
Readable
Runnable
Thread.UncaughtExcept
Classes
Boolean
Byte
Character
Character.Subset
Character.UnicodeBlock
Class
ClassLoader
ClassValue
OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
Java™ Platform Standard Ed. 8
PREV NEXT FRAMES NO FRAMES
Java™ Platform, Standard Edition 8
API Specification
This document is the API specification for the Java™ Platform, Standard Edition.
See: Description
Profiles
compact1
compact2
compact3
Packages
Package
Description
java.applet
Provides the classes necessary to cre
an applet and the classes an applet u
to communicate with its applet conte
java.awt
Contains all of the classes for creatin
user interfaces and for painting grap
and images.
java.awt.color
Provides classes for color spaces.

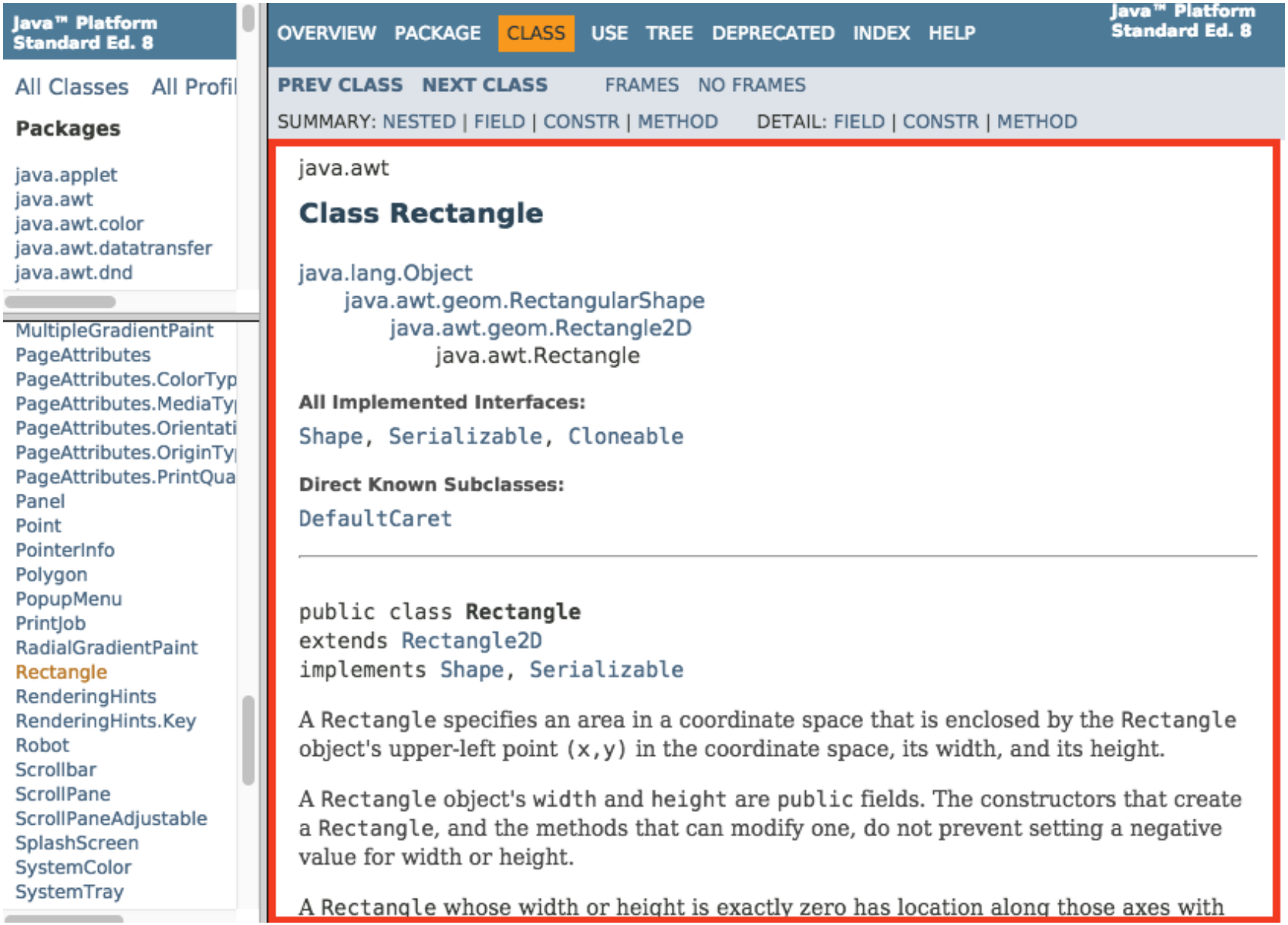
List of packages



List of classes in package



Detail for a selected class



Class documentation

The API documentation for each class contains;

- A description of the functionality of the class
- A table of the instance variables in the class
- A table of the constructors in the class
- A table of the what methods in the class
- A detailed **description** of each constructor and method

Using a new class

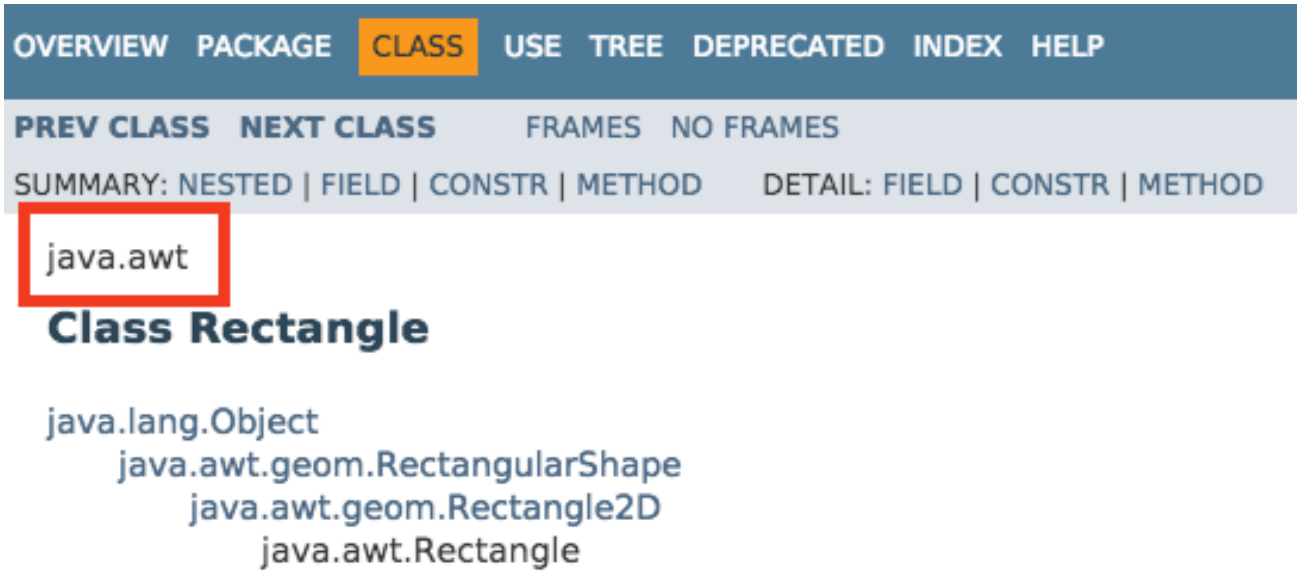
There are a number of steps required before we can use a class from the library

1. We have to **import** the class into our code
2. We have to **construct** an object based on the class
3. We have to look up the descriptions of its methods

Using a new class

1. Importing a new class

First we find what package the class is saved in. This information is at the top of the class documentation



Syntax

The syntax is `import package.className;`

e.g. `import java.awt.Rectangle;`

Class Documentation

Method and constructor definitions

The API contains a description for each of the constructors that we can use to create an object

Constructors

Constructor and Description
`Rectangle()` Constructs a new `Rectangle` whose upper-left corner is at (0, 0) in the coordinate space, and whose width and height are both zero.
`Rectangle(Dimension d)` Constructs a new `Rectangle` whose top left corner is (0, 0) and whose width and height are specified by the `Dimension` argument.
`Rectangle(int width, int height)` Constructs a new `Rectangle` whose upper-left corner is at (0, 0) in the coordinate space, and whose width and height are specified by the arguments of the same name.
`Rectangle(int x, int y, int width, int height)` Constructs a new `Rectangle` whose upper-left corner is specified as `(x,y)` and whose width and height are specified by the arguments of the same name.

Class Documentation

2. Constructing a new class

Once we have chosen a constructor to use we need to find the parameters required

Rectangle(int x, int y, int width, int height)

Constructs a new Rectangle whose upper-left corner is specified as (x,y) and whose width and height are specified by the arguments of the same name.

We can see that the constructor requires 4 int parameters to construct a `Rectangle` object

Example

```
Rectangle r = new Rectangle(5, 5, 10, 20);
```

This creates a `Rectangle` object at the position (5, 5) with a width of 10 and height of 20

Class documentation

3. Using the object

To use the object we need to know what its methods do. This information is in the documentation

contains

```
public boolean contains(int x,
                        int y)
```

Checks whether or not this Rectangle contains the point at the specified location (x,y).

Example

```
Rectangle r = new Rectangle(5, 5, 10, 20);
boolean ans = r.contains(10, 10);
```