Classes and objects, part 2

02

- Defining classes
- Methods and fields
- Constructors
- Access control

Classes

- Java is a *class-based* language.
- Every object is an instance of a class.
  - The class is the object's type.
  - The class is a "template" for objects of that type.
- Java allows the programmer to define new classes, so the set of object types is unlimited.

Defining a class

```
public class Length {
  // fields
  int yards;
  int feet;
  int inches;
  // constructors
  public Length(int yd, int ft, int in) { .. }
  public Length(Length len) { ... }
  // methods
  public void add(Length len) { ... }
  public int toInches() { ... }
  public double toMeters() { ... }
}
```

- A class definition consists of:
  - (optional) access modifier, e.g. `public`;
  - keyword `class`;
  - the class name; and
  - class body in braces (“`{ }`”), with:
    - field declarations;
    - constructor declarations; and
    - method declarations.
- All items in the class body are optional.
- A class can have both static and instance methods.

Fields

- Fields define the object state.
  - Like variable declaration, can be initialised.
- Instance (non-static) methods can depend on, and modify, the object's fields, in addition to arguments.
- In an instance method, refer to fields of the object that the method is called on by their name only.
- Reference fields in another object:

```
len.yards * 3 + len.feet
```

  - Expression that evaluates to an object; dot (“.”); field name.

Constructors

- A *constructor* is like an instance method, but has no return type and its name is the class name.
- To create a new object (instance of a class):

```
Length height = new Length(0, 6, 3);
```

- The matching constructor is called when the object is created; its purpose is to initialise the new object.
- A class can have several constructors with different parameters (“overloading”, or compile-time *polymorphism*).

Keyword `this`

- Within an instance method (or constructor), the keyword `this` is an expression that evaluates to the object that method is called on:

```
this.yards = len.yards;
```

- Within a constructor only, `this` can be used to call a different constructor of the class:

```
public Length(Length len) {
  this(len.yards, len.feet, len.inches);
}
```

Access restrictions

- Access modifier keywords can be added to class, field and method declarations:
  - `private`: accessible within the class only.
  - (`protected`: within class and its subclasses.)
  - No modifier: within the class' package.
  - `public`: accessible everywhere.
- Access restrictions forces code to interact with objects only through their intended "interface", and thus promotes *abstraction*.

Example

- Change `Length`'s internal representation to mm, keeping the interface (public methods) the same.

Common object methods

- Certain methods are defined for every Java object:
  - Every type has a method `String toString()` that creates a "printable" string representation of values of the type. This is invoked automatically in certain cases.
  - Every class type has a method `boolean equals(Object)` that defines class-specific object equality.
- When creating a class, we can define *what* these methods do.
- If we don't, they will have default behaviour.

preview (O4): Inheritance

- Java classes form a hierarchy:
  - sub-classes extend their super-class.
- `java.lang.Object` is the root of the class hierarchy (ultimate ancestor of every class).
  - Methods defined by `Object` are inherited (if not overridden) by every class.