

## Outline: Class implementation

There are actually a number of new ideas in classes that we know even when using them. We want to learn them all.

- Define a class, i.e., a new type in Java.
- Define the instance fields of the class, i.e., reserve space for **each** instance, and specify what it should hold.
- Define the instance methods of the class, i.e., provide some methods for accessing an instance so that users don't need to know the internals of the class.
- Specify that some of the fields or methods is "internal", i.e., the users should not call such methods or read/modify such fields. This can be used to establish some **data invariant** in the object.

We will try to learn them one by one.

0911B L19-1

## Defining a class

- When we say "public class XXX { ... }", we are actually defining a class. The name of the class is the XXX after the keyword class.
- There can only be one public class in one Java file. However, one Java file can have many class'es, just drop the keyword public.
- That is something called an access specifier, and we will learn them later. Currently, just use public for the main program, and drop public for anything else.
- It should be clear now that even our main program is itself a class. When we write a "program", we are actually writing a class that contains a static main method—one that can be called by running the Java bytecode interpreter directly.
- However, so far our classes contain no data fields.

0911B L19-2

## Adding data fields

- A class has few uses if it does not contain a data field.
- A data field specify the "content" of an object of that particular class. Each object has one copy of each of the data fields. Intuitively, the object group all the field form to a single variable.
- Data fields are created just like class variables: they are defined outside all methods of a class. The only difference is that we replace the word **static** by **public**.
- So to define a class containing a String field "name" and an int field "age", we need `public String name;` and `public int age;`
- Initially, a field contains something like 0, 0.0, false, etc.

0911B L19-3

## Example: A class for playing cards

A playing card contains two things: the suit, and the value. Let us represent them by two bytes. For suit, 0, 1, 2 and 3 represents Spade, Heart, Diamond and Club, and for number, 2 to 14 represents the values 2, 3, ..., 10, Jack, Queen, King and Ace.

```
class PlayingCard {
    public byte suit; // 0 to 3 represents Spade, Heart, Diamond and Club
    public byte value; // 2 to 14 represents 2, 3, ..., 10, J, Q, K and A
}
```

Now whenever we create a PlayingCard object, we have something like the following:

A PlayingCard object (Club 9) 

(byte)suit:	3
(byte)value:	9

Note: it makes sense to talk about the suit of a PlayingCard object, but it doesn't make sense to talk about the suit of the program.

0911B L19-4

## Using the resulting object

There are two things that we can do with the object now: to create it (using `new PlayingCard()`), and to assign values into its fields.

Given a reference `card` to `PlayingCard`, we can access its "suit" field by `card.suit`, and its "value" field by `card.value`.

For example, the following program create the object in the last page:

```
public class CreateClub {
    public static void main(String[] args) {
        PlayingCard card = new PlayingCard();
        card.suit = 3;
        card.value = 9;
    }
}
```

Notice that it does not make sense to write `suit` unless you give a reference to a `PlayingCard` object before it: "suit" is a property of a `PlayingCard`.

0911B L19-5

## One more example

One more example: to create all the cards of a deck:

```
public class CreateDeck {
    public static void main(String[] args) {
        // Create an array of PlayingCard references
        PlayingCard card[] = new PlayingCard[52];
        int curr = 0;
        for (int i = 0; i < 3; ++i) {
            for (int j = 2; j <= 14; ++j) {
                card[curr] = new PlayingCard(); // Really create a playing card
                card[curr].suit = (byte) i;
                card[curr].value = (byte) j;
                ++curr;
            }
        }
    }
}
```

0911B L19-6

### **The null reference**

- In line 4 of the last program, we create an array of `PlayingCard` references. We actually create `PlayingCard` objects for them at line 8. What the `PlayingCard` references hold before line 8 is executed?
- The answer: they hold `(PlayingCard)null`, meaning “does not point to an object”. Whenever you have a class variable or array element of a reference type, it holds `null` initially (unless specified otherwise).
- Of course, you cannot use the object pointed to by a `null` reference: there is no such object. It gives you a run-time error.
- Things like `card[i]=null;` and `card[i]==null` work as expected.
- It is very useful in constructing advanced data structures.