



THE UNIVERSITY OF
MELBOURNE

Workshop Week5

COMP90041 Programming
and software development

Zhe(Zoe) Wang





Methods

- **static** methods
- **instance** methods/ **non-static** methods

static methods

call methods (in another class)

- Form: `Class.method(expr1, expr2, ...)`
- The `exprs`, called arguments, provide data for the method to use

- General form (for now):

```
public static type name(type1 var1, type2 var2, ...) {  
    :  
}
```

- Each `var` is called a parameter
- Then body (`:` part) of method is executed
- Types of corresponding arguments and parameters must match
- `type` is type of result returned

void

- `main` is a class method we've been defining
- Java executes the `main` method when running an application
- Begins with:

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

- Ends with:

```
}
```

headers and Signatures

- First part of method definition (up to `{`) is called the method header
- Method name plus number and types of arguments together are called the method signature

```
public static int calInt(int num1, int num2){  
    return num1 + num2;  
}
```

header

signature

overloading

- Overloading: when a method name has multiple definitions, each with different signature
- Java automatically selects the method whose signature matches the call
- You cannot overload based on return type, only parameter types

wrong!

```
int    bad(int x, double y) {...}  
double bad(double x, int y) {...}
```

defining constants

- Form:

```
public static final type name = value;
```

- E.g.:

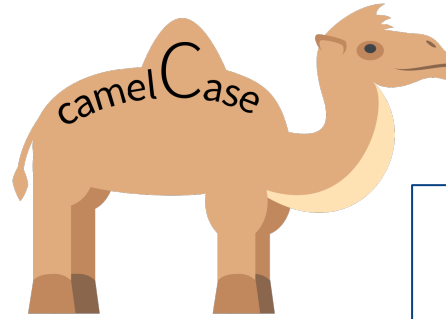
```
public static final int DAYS_PER_WEEK = 7;  
public static final int CARDS_PER_SUIT = 13;
```

- Best practice: don't sprinkle mysterious numbers in your code, define constants instead

price = appleNumber *2

magic number!

Variable Name Rules



- Variable names begin with a letter, and follow with letters, digits, and underscores (_)

- Begin with **lower** case letter

- Follow with lower case, except: **Capitalize** first letter of each word in phrase

- Make it **descriptive**

helloWorld

A

a
b
c

B

boxLength
boxWidth
boxHeight

C

1box
2box
3box

D

BoxLength
BoxWidth
BoxHeight

local variable

demo2

- Variables declared inside methods are local to the method (cannot be used outside)
- Local variables cannot be declared `public` or `private`

class variables (static)

- Class variable is a variable that is local to a class
- Can be declared either `public` or `private`
- It should almost always be `private`
 - ▶ Difficult to control if every method can modify it

instance variable

- ▶ Instance variables, which hold the data of an object

Form: `private type name;`

```
public class Person {  
    private String familyName;  
    private String givenName;  
    :  
}
```

- Local variables live in a method; class variables live in a class; instance variables live in an object

instance(non-static) methods

- ▶ (Instance) methods, which define the operations (code) of an object

Call a static method

```
SampleClass.method1();
```

Call a non-static method

```
SampleClass myObject = new SampleClass();  
myObject.method2;
```

objects

- Each object is an instance of some class
template!

demo2

A class holds operations and data related to one concept.

creating objects



- When creating an object, its instance variables need to be initialised to appropriate values
- Constructors are special methods responsible for this

```
public ClassName(type1 var1,...) {  
    :  
}
```

default constructors?

```
Classname myObject = new Classname(...);
```

```
public class SampleClass {  
    public static void method1(){  
        method2();  
    }  
  
    public void method2(){  
        method1();  
    }  
}
```



toString method

dog demo

if p is a Person object

- What should `System.out.print(p)` print?
- Define a public method `String toString()`

```
public String toString() {  
    return givenName + " " + familyName;  
}
```

accessor/getter

Mutator/setter

```
public class Person {  
    private String name; // private = restricted access  
  
    // Getter  
    public String getName() {  
        return name;  
    }  
  
    // Setter  
    public void setName(String newName) {  
        this.name = newName;  
    }  
}
```



Q1

Write a class named Appointment that contains instance variables **startTime**, **endTime**, and a date which consists of a day, month (valid values are January through December) and year. All times should be in military (24 hours) format, therefore it is appropriate to use integers to represent the time. Write accessor and mutator methods, and helper methods for setting and displaying an appointment.



THE UNIVERSITY OF
MELBOURNE

Thank you
