Object Oriented Programming in R

Song Liu (song.liu@bristol.ac.uk)
GA 18, Fry Building,
Microsoft Teams (search "song liu").

In today's lecture, we will learn how to do OOP in R by

- Creating objects,
- Writing functions for objects,
- Inheriting from a parent class.

Object Oriented Programming (OOP) (Revision)

- In OOP, your code is divided into small parts called **objects**.
  - These parts can have hierarchies reflecting the real-world relationship between objects.
  - If an object is a `CSstudent`, then it is a `student`.
- Objects contain data **as well as procedures that operates on the data**.
  - Solves the "data-operation detachment" issue.
  - The **procedures** in an object are called "**methods**".
  - The **data** in an object are called "**fields**".

OOP in R

- R is an object oriented programming language.
  - It allows you to create **objects**, inherit **classes** and write functions for objects in specific classes.
  - However, OOP in R is quite different from other classic OOP languages, such as C++ or Java.
- There are three different OOP systems in R.
  - S3, the only system we care in this unit.
  - S4
  - Reference

Class

- A class groups related variables as well as procedures together in one entity.

```
#include <stdio.h>
class student{
    int ID;
    char* name;
    int grade;
};
```

- Then we can create objects using the class definition.

```
student song, jack;
```

Objects in R

- In R, we do not need to write the formal class definition when creating objects.
- Since an object is essentially the group of revelant variables and methods, we can use a list to combine everything together.

```
song <- list(name = "song liu", ID = 1234, grade = 70)
jack <- list(name = "jack jones", ID = 2345, grade = 80)
```

- Then we tell R, they are objects from the `student` class.

```
class(song) <- "student"
class(jack) <- "student"
```

Objects in R

- Now R thinks both `song` and `jack` are objects of `student` class.

```
> class(song) # class function returns class name
# of the input
[1] "student"
```

- In fact, all data structures, functions in R are objects:

```
dataset <- data.frame(1:4, 2:5)
class(dataset)
[1] "data.frame" #a data frame is an object of
# "data.frame" class

class(print)
[1] "function" #a function is an object of "function"
# class
```

Be Careful with `class`

- The object construction in R is surprisingly informal comparing to the OOP system in C++! E.g., you can do:

```
cube <- list(height = 70, width = 150, depth = 50)
class(cube) <- "student"

class(cube)
[1] "student"
```

  - This makes no sense!

- Since there is no class definition, R has ***no way*** to know whether an object is an instance of its class or not.

Be Careful with `class`

- It is very dangerous to change class for the varibles of builtin classes.

```
dataset <- data.frame(1:4, 2:5)
> dataset[1,] # it works!
    X1.4 X2.5
1    1    2

class(dataset) <- "student"
dataset[1,] # doesn't work!
Error in dataset[1, ] : incorrect number of dimensions
```

Inside an Object

- We can check out the contents of an object using `print`

```
print(song)
$name
[1] "song liu"

$ID
[1] 1234

$grade
[1] 70

attr(,"class")
[1] "student"
```

- Not surprisingly, an object in R is essentially a list.
    - `attr(,"class")` is an attribute, which provides additioanl information for a data structure.

Methods

- Methods are functions that are "attached" to an object.

```
class student{
public:
    int ID;
    char* name;
    int grade;
    void print(){
        printf("%d", grade);
    }
};
```

- We can call these methods using `.` syntax.

```
student song;
song.print();
```

Methods in R

- In R, we cannot write a function "attached" to a class, as there is **no** class definition.
- However, you can always write function that operates on objects from specific classes.

```
print_grade <- function(s){
    print(s$grade) # all variables of an R object are
    # visible from outside.
}
```

- Be careful, it does not check for "class correctness":

```
cube <- list(height = 70, width = 150, depth = 50)
class(cube) <- "student"

print_grade(cube)
NULL # there is no "grade" in cube!
```

Polymorphism

- Polymorphism in computer science means "a single interface for different data types"
- Polymorphism in R means the same function will behave differently for objects from different classes.

Polymorphism

- Notice `print` function behaves differently when printing a list and a vector?

```
print(list(1:4))
[[1]]
[1] 1 2 3 4

print(1:4)
[1] 1 2 3 4
```

- R uses different versions of `print` when the inputs are objects from different classes!

Polymorphism

- Now try to print our `student` objects:

```
print(song)
> print(song)
$name
[1] "song liu"

$ID
[1] 1234

$grade
[1] 70

attr(,"class")
[1] "student"
```

- No surprise, cannot find a specific implementation for `student` class, R treats our `song` as a list.

Polymorphism

- However, can we tell R to recognize our `student` class and behave correspondingly?
- We can write a function `print.student`

```
print.student <- function(s){
    print(paste(s$name,
          "(ID: ", s$ID, ") has a score", s$grade))
}
```

- Now, R will run `print.student(s)` when we call `print(s)`.

```
> print(song)
[1] "song liu ( ID: 1234 ) has a score 70"
```

Polymorphism

- Suppose `input` is an object of class `c`.
- If the definition of `func.c` is provided, R will call `func.c(input)` when you write `func(input)`.

Inheritance

- Inheritance is an important concept in OOP and models the hierarchical relationship among real-world objects.
- A child class inherits from its parent **reuses all code** written for the parents class.

Inheritance

- In R, inheritance is done by using `class` function.

```
song <- list(name = "song liu", ID = 1234, grade = 70,
             country = "China")
jack <- list(name = "jack jones", ID = 2345, grade = 80)

class(jack) <- "student"
class(song) <- c("international_student", "student")
```

- `class(obj) <- c("child", "parent")` tells R that `obj` is an object of class `child` that **inherits** class `parent`.
- Now, R knows `song` is a `student` thus inherits all code written for `student`.

```
print(song)
[1] "song liu (ID: 1234 ) has a score 70"
```

Inheritance

- We can also write a `print` function for `internationa_student` so it behaves differently when its input is an `international_student`.

```
print.international_student <- function(s){
    print(paste(s$name,
        "(ID: ", s$ID, ") has a score", s$grade,
        "and is from", s$country))
}
```

```
print(song)
[1] "song liu (ID:  1234 ) has a score 70 and is from China"
print(jack)
[1] "jack jones (ID: 2345 ) has a score 80"
```

Conclusion

R programming language supports OOP.

- R does not allow the explicit definition of a class.
- An object is created from a list by using `class` function.
- The same function call can lead to different behavior when the input objects are from different classes.
- Inheritance allows you to reuse methods written for the parent classes.