Congratulations, you sank my battleship!

Object oriented programming in R

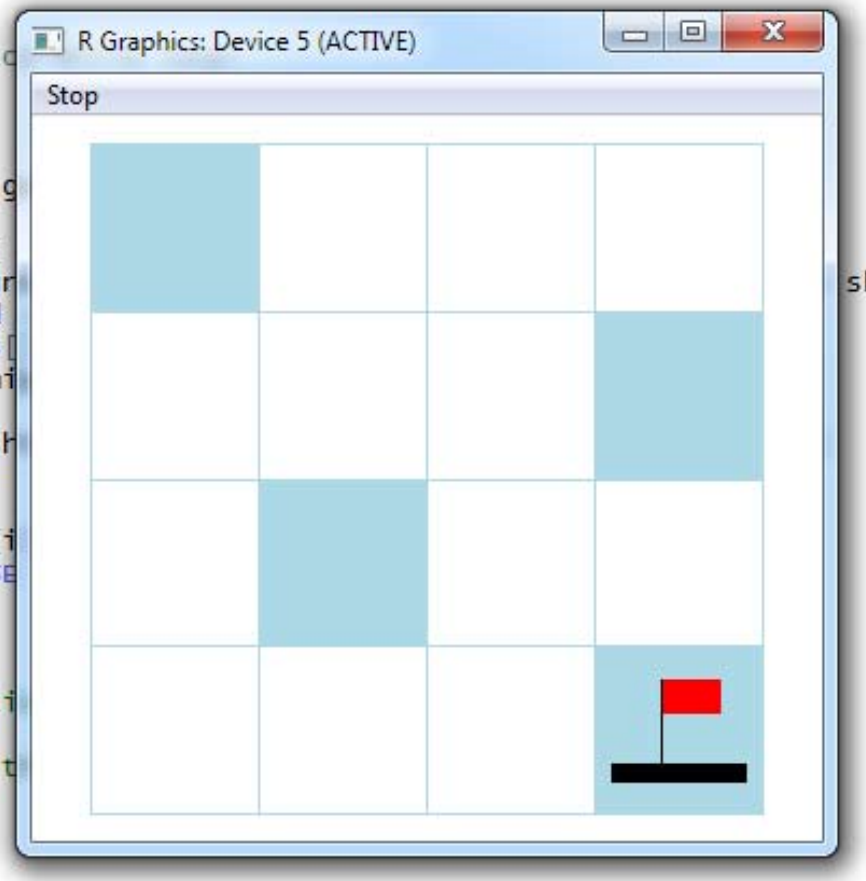
Janina Torbecke
Inga Schwabe

5^{th} TRUG meeting
January 16, 2014

Outline

- OOP
- OOP in R
- Battleship
- Advantages & disadvantages of OOP in R
- Other applications

```
}

allFound <- TRUE #Sto

found <- FALSE

plot.water(guess$x, g

for(i in 1:nships) {
  if (ships[[i]]@four
    ships[[i]]@found
    plot.water(ships[
    plot.ship_img(shi
    # Good guess:
    if (guess$x == sh
      found <- TRUE
    }
  }else if (!ships[[i
    allFound <- FALSE
  }
}
if (found) {
  print("Congratulati
}else{
  print("You missed t
}

}
}
```



Object oriented programming (OOP)

- Objects

  which

  - have attributes
  - have associated procedures (methods)
  - are usually instances of classes

For example..

- A student, member of the class "Students" with attributes name, age, grade
- For different objects different methods
  (e.g. generic function "plot", different procedures for different objects)

OOP in R

1. S3
2. S4
3. R5

The S4 system

Define an S4 class with `setClass()`:

```
1 #Class Student:
2 setClass(Class = "Student",
3 representation(fullname = "character", age = "numeric"),
4 prototype(fullname = NA_character_, age = NA_real_))
```

Class	The name of the class
Representation	A named list of the slots (= class attributes) indicating the class of each slot
Prototype	An object with default data for the slot
Contains	Names of the superclasses (For inheritance purposes, explained later)

The S4 system

Create an instance/object of the class with `new()`:

```
#New instance of the class "Student":
Gerd <- new("Student", fullname = "Gerd Jansen", age = 21)
Jan <- new("Student", fullname = "Jan van der Meulen", age = 19)
```

Access class slot by `@` operator:

```
Gerd@fullname #Prints "Gerd Jansen"
Gerd@age #Prints "21"
```

Methods and Generic Functions

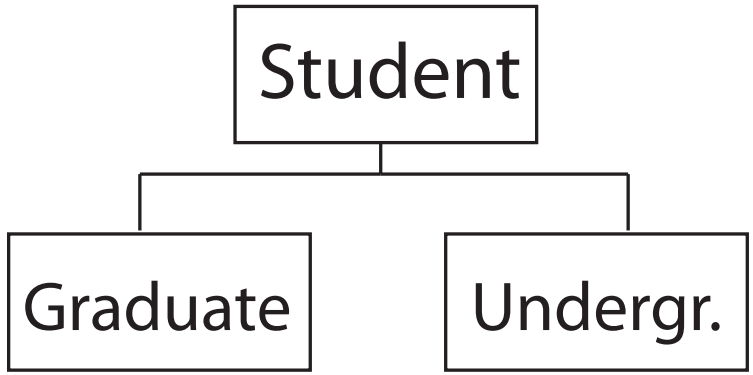
- Generic functions alllow different methods to be selected corresponding to the classes of the objects supplied as an argument in a call to the function

```
#Set your own generic function (e.g. How many sides has a shape?)
setGeneric(name = "sides", def = function(object){
         standardGeneric("sides")}, valueClass = "character")

#standarGeneric() dispatches the method defined for a generic function

#Define different methods
setMethod("sides", signature("Triangle"), function(object) return("3"))
setMethod("sides", signature("Circle"), function(object) return("Infinite"))

> sides(new("Triangle"))
[1] "3"
> sides(new("Circle"))
[1] "Infinite"
```

Inheritance

- An object or class is based on another object or class
- Example: Graduate students & undergraduate students with partly same behavior

```
1 #Define superclass (other ways possible)
2 setIs("Undergraduate", "Student")
3
4 #Check relationship
5 extends("Undergraduate", "Student")
```



- Inheritance of methods

```
1 Use callNextMethod() in setMethod()
```

Battleship: An application of the S4 system

```
1 #New class "Ship"
2 setClass("Ship", representation(posX = "numeric",
3 posY = "numeric", found = "logical"), prototype(found = FALSE))
4
5 #New Ship:
6 ships <- list()
7 ships[1] <- new("Ship", posX = sample(1:nRow,1),
8       posY = sample(1:nCol,1))
```

Battleship: An application of the S4 system

```
1   for(i in 2:nships) {
2     foundPos <- FALSE
3     shipRow <- NA
4     shipCol <- NA
5
6     while(!foundPos){ #Loop continues until position of 1 ship is found
7       shipRow <- sample(1:nRow,1)  #Random row nr
8       shipCol <- sample(1:nCol,1)  #Random col nr
9
10      #Check if position is available
11      for(j in 1:length(ships)) { #Iterate over all ships in the list
12        foundPos <- !(ships[[j]]@posX == shipRow && ships[[j]]@posY == shipCol)
13        if(!foundPos) {
14          break
15        }
16      }
17    }
18    #New ship on available position
19    ships[i] <- new("Ship", posX = shipRow, posY = shipCol)
20  }
```

Advantages & disadvantages of OOP in R

Advantages:

- Enables the use of generic functions
  (necessary in order to build your own R library)
- Efficiently programming, e.g. elimination of redundant code through inheritance
- Neat code

Disadvantages:

- Not always a big difference between OOP & functional programming
- Execution time may increase

Other (possible) applications

- School data
- Modelling of social networks
- Cognitive psychology (e.g. neurons in neural network)
- ...

S4 Key functions

`setClass()`	Create new class
`setIs()`	Define superclass of a class
`extends()`	Check relationship(s) between classes
`isClass()`	Check class name
`is()`	Show all subclasses of a class
`getClasses`	Show all classes of an object
`removeClass()`	Remove class
`@/slot()`	Access slots
`slotNames()`	Show slot names
`getSlots()`	Show slot names + their classes
`setGeneric()`	Create new generic function
`setMethod()`	Define new method
`methods()`	Show all methods of a generic function