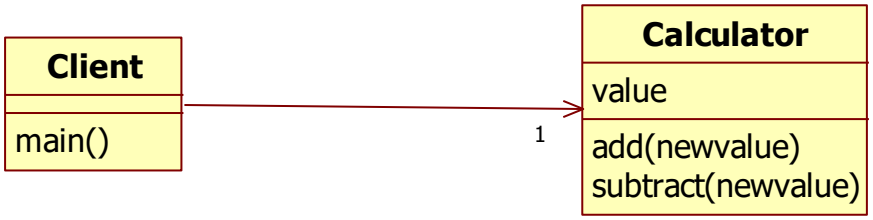
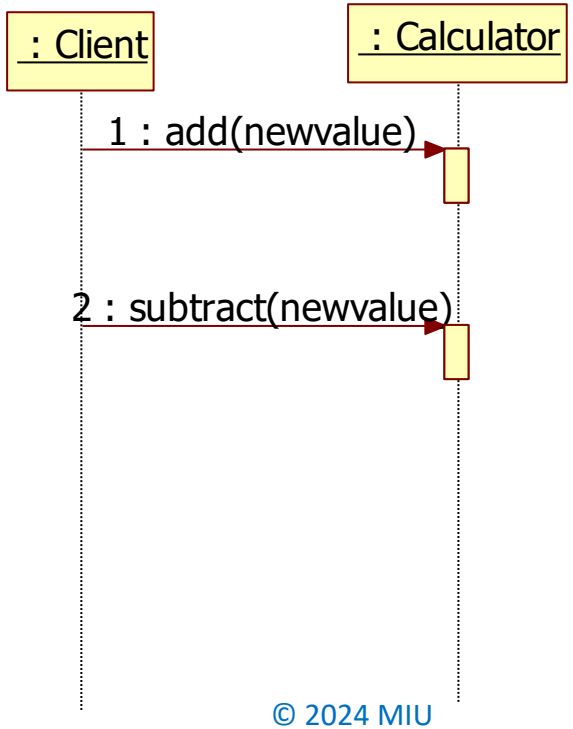
LESSON 2
COMMAND PATTERN

Command pattern

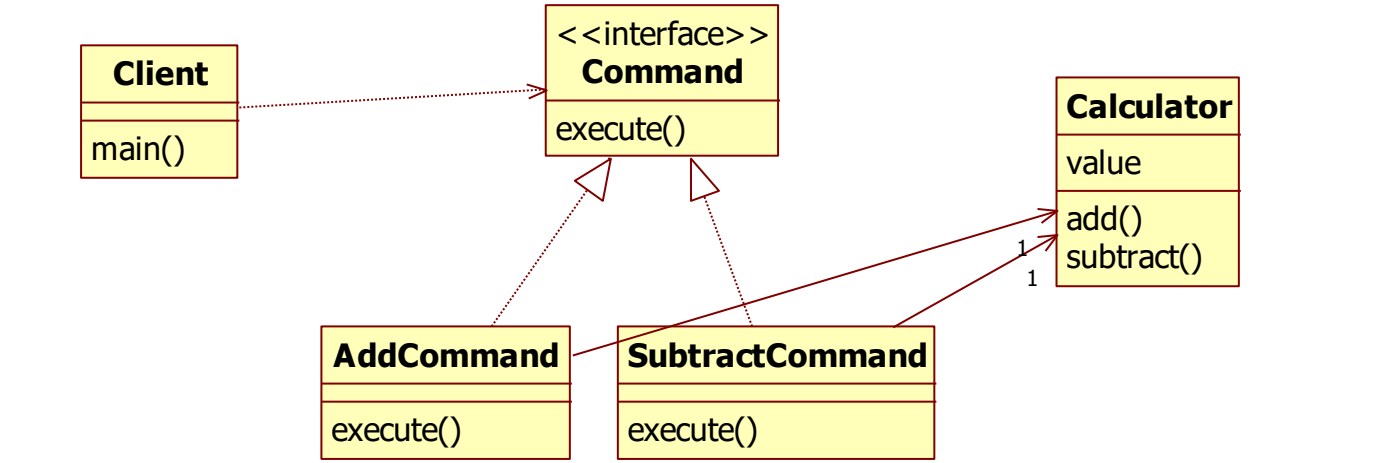
- Encapsulate a request into a single object
- Advantages:
  - Command objects can be logged
  - Command objects can be used for undo/redo functionality
  - Command objects can be replayed

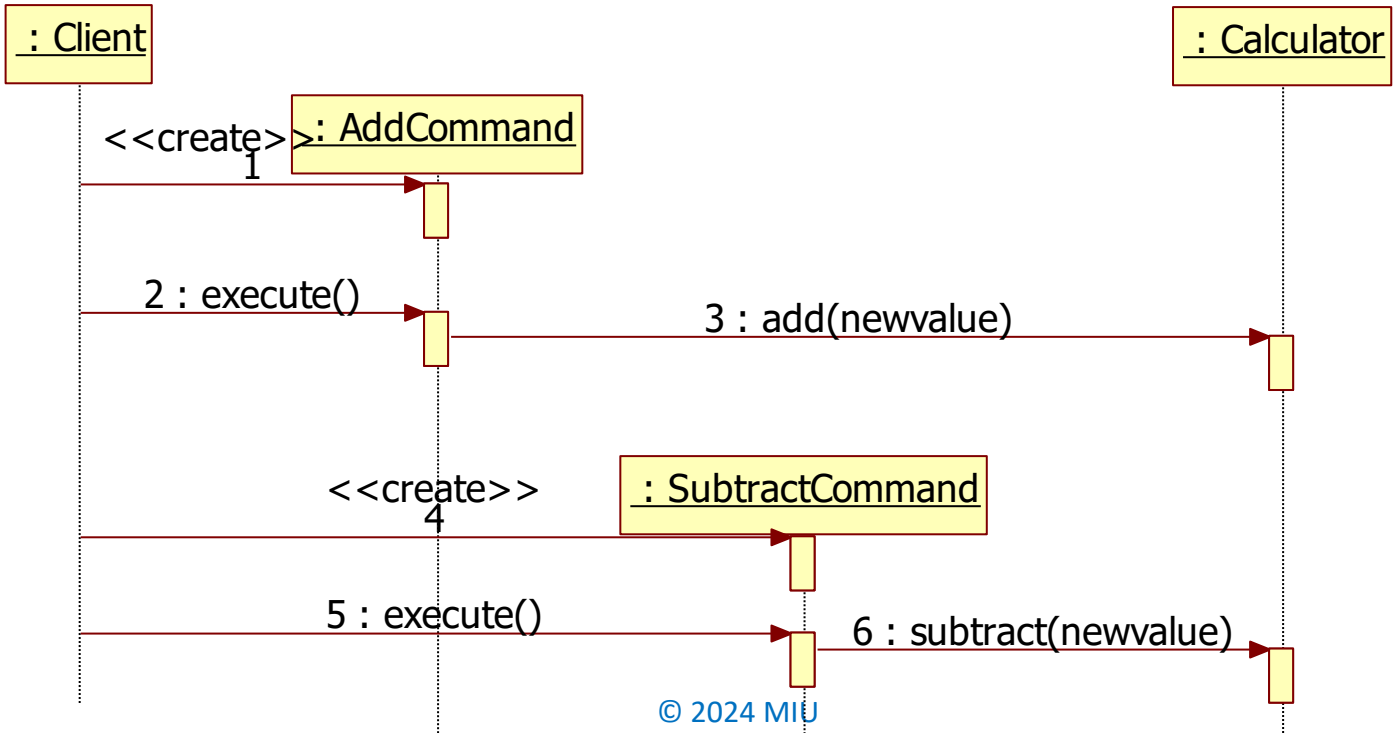
Without Command pattern



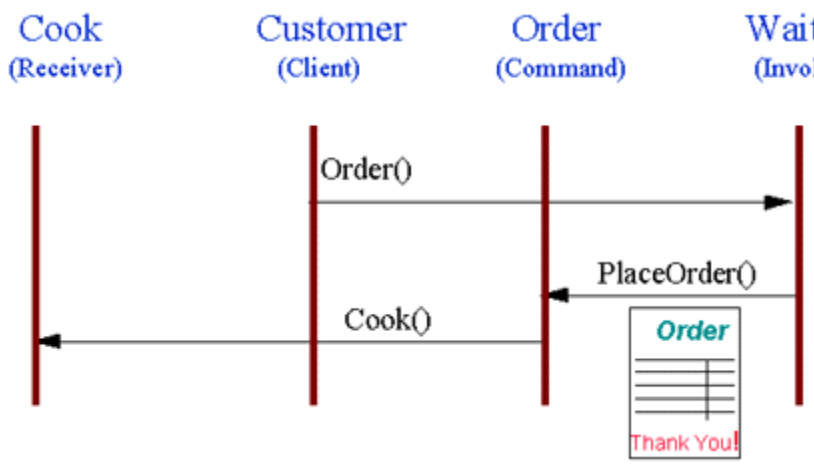


With Command pattern



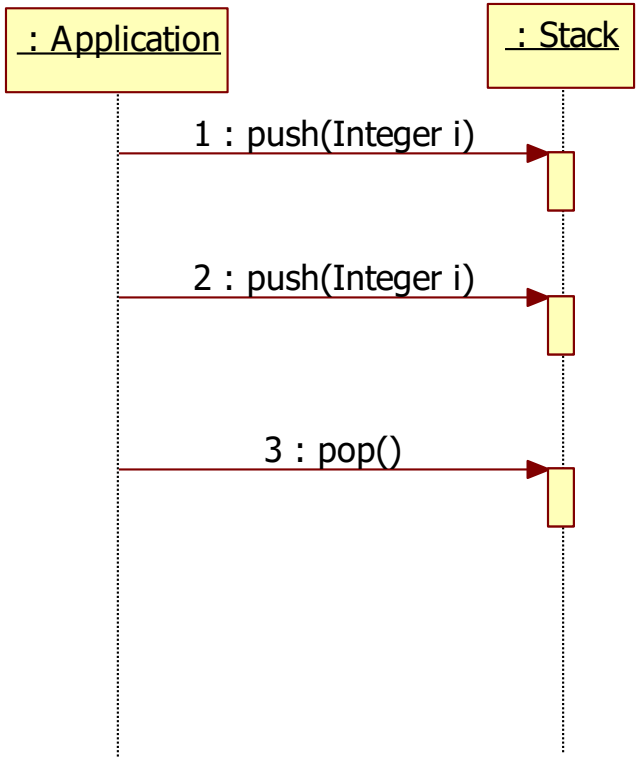


Non software example



- The waiter or waitress takes an order, or command from a customer, and encapsulates that order by writing it on the check. The order is then queued till the cook has time to work on the order.

Example without command



Stack without command

```
public class Stack {
  private List<Integer> list = new ArrayList();

  public Integer pop(){
    Integer top = null;
    Iterator iter = list.iterator();
    while (iter.hasNext()){
      top = (Integer)iter.next();
    }
    iter.remove();
    printStack();
    return top;
  }

  public void push(Integer value){
    list.add(value);
    printStack();
  }

  public void printStack(){
    System.out.println("current stack ------");
    for (Integer i : list){
      System.out.println("--"+i);
    }
    System.out.println("end of stack ------");
  }
}
```

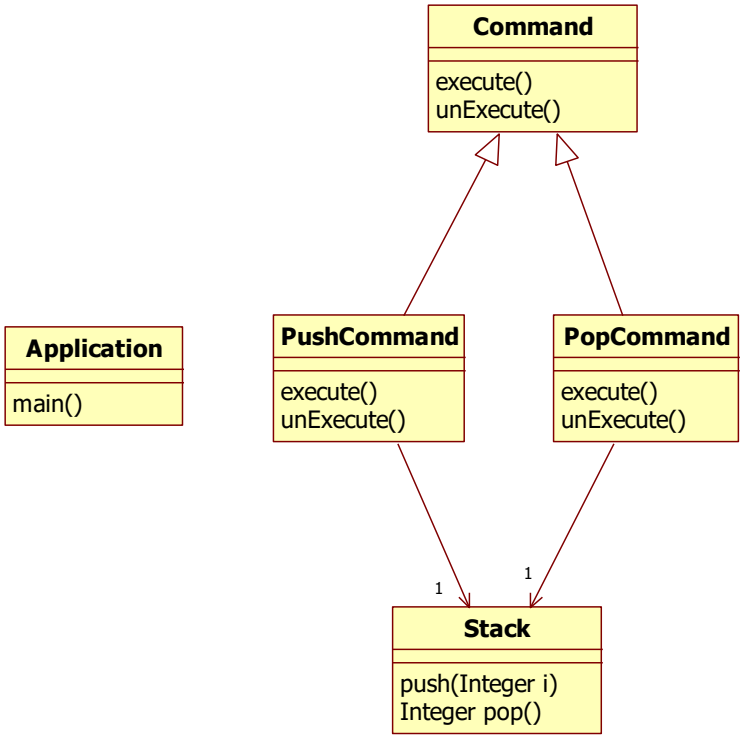
Stack without command

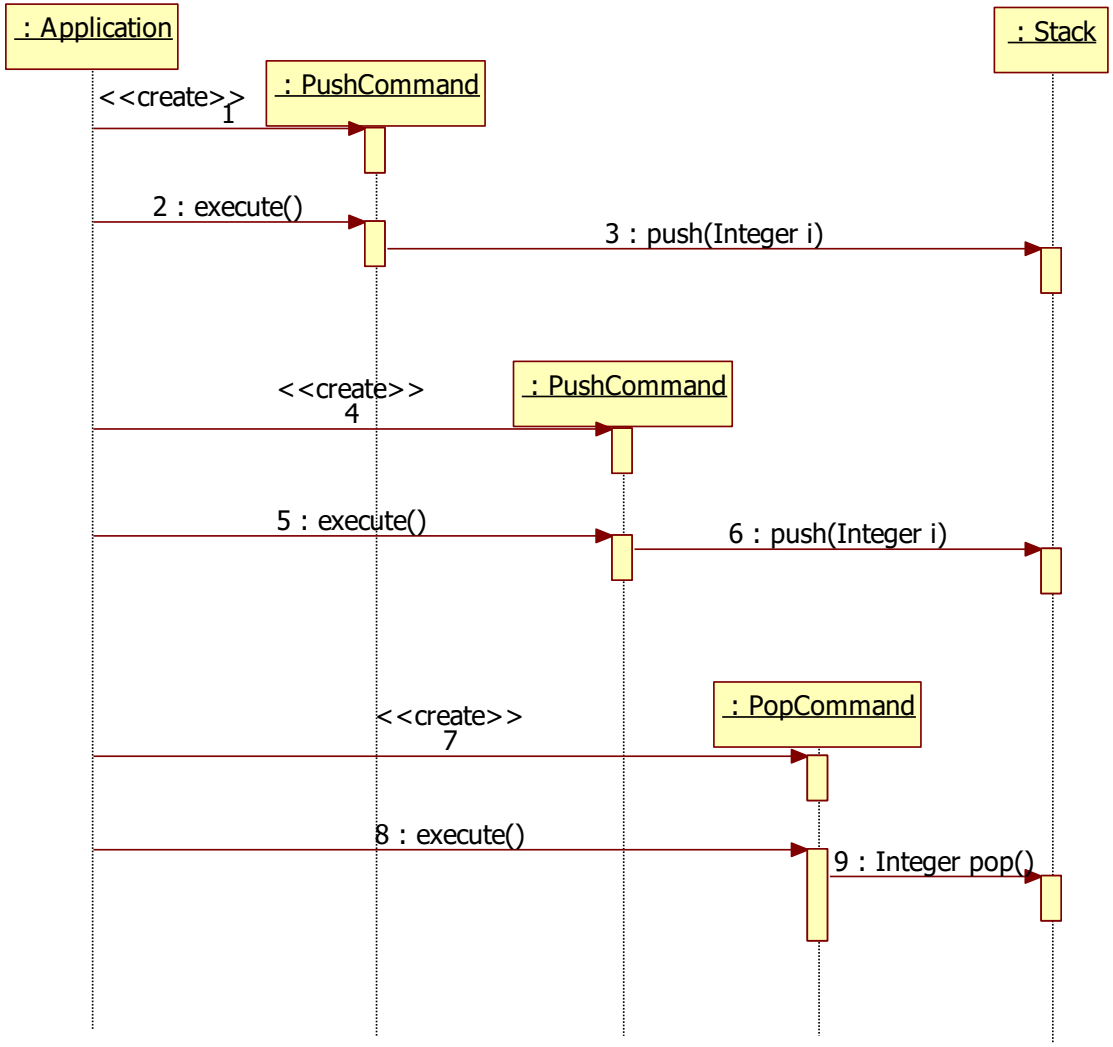
```
public class Application {

  public static void main(String[] args){
    Stack stack = new Stack();
    stack.push(new Integer(6));
    stack.push(new Integer(2));
    stack.push(new Integer(8));
    System.out.println(stack.pop());
    System.out.println(stack.pop());
    System.out.println(stack.pop());
  }
}
```

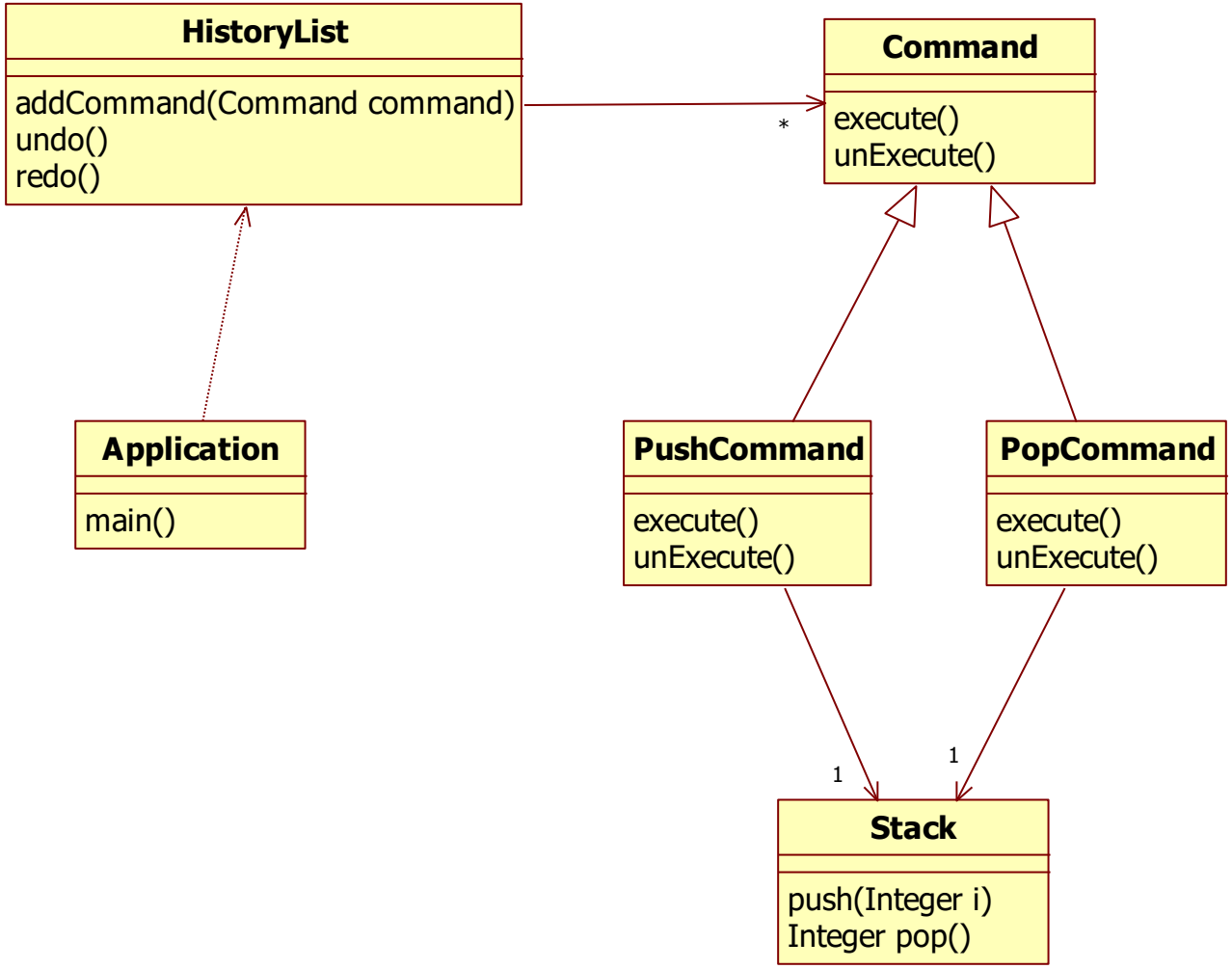
```
current stack ------
--6
end of stack ------
current stack ------
--6
--2
end of stack ------
current stack ------
--6
--2
--8
end of stack ------
current stack ------
--6
--2
end of stack ------
8
current stack ------
--6
end of stack ------
2
current stack ------
end of stack ------
6
```

Stack with command

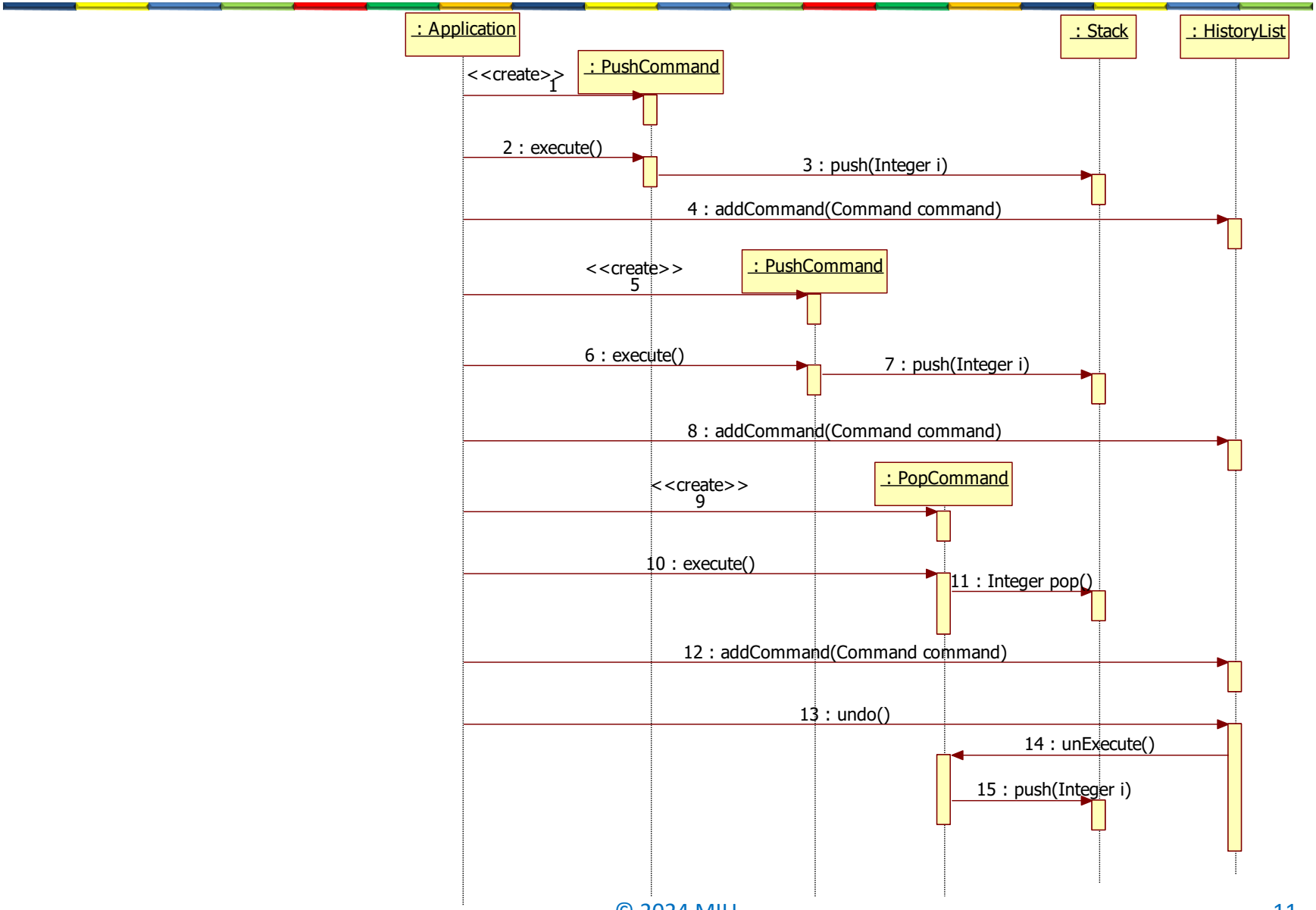




Undo/redo functionality



undo/redo functionality



Stack with command

```
public interface Command {
  void execute();
  void unExecute();
}
```

```
public class PushCommand implements Command{
  Stack stack;
  Integer i;


  public PushCommand(Stack stack, Integer i) {
    this.stack = stack;
    this.i=i;
  }

  public void execute(){
    stack.push(i);
  }

  public void unExecute(){
    stack.pop();
  }
}
```

```
public class PopCommand implements Command{
  Stack stack;
  Integer i;


  public PopCommand(Stack stack) {
    this.stack = stack;
  }

  public void execute(){
    i=stack.pop();
  }

  public void unExecute(){
    stack.push(i);
  }
}
```

HistoryList

```
public class HistoryList {
  private Collection<Command> commandlist = new ArrayList<Command>();
  private Collection<Command> undolist = new ArrayList<Command>();

  public void undo() {
    if (commandlist.size() > 0) {
      Command commandObject = (Command) ((ArrayList<Command>) commandlist).get(commandlist.size() - 1);
      ((ArrayList<Command>) commandlist).remove(commandObject);
      commandObject.unExecute();
      undolist.add(commandObject);
    }
  }

  public void redo() {
    if (undolist.size() > 0) {
      Command commandObject = (Command) ((ArrayList<Command>) undolist).get(undolist.size() - 1);
      commandObject.execute();
      ((ArrayList<Command>) undolist).remove(commandObject);
      commandlist.add(commandObject);
    }
  }

  public void addCommand(Command commandObject) {
    commandlist.add(commandObject);
  }
}
```

Application

```
public class Application {

  public static void main(String[] args) {
      Stack stack = new Stack();
      HistoryList hlist = new HistoryList();
      PushCommand pushc1 = new PushCommand(stack, new Integer(6));
      pushc1.execute();
      hlist.addCommand(pushc1);
      System.out.println(stack);

      PushCommand pushc2 = new PushCommand(stack, new Integer(3));
      pushc2.execute();
      hlist.addCommand(pushc2);
      System.out.println(stack);

      PopCommand popc1 = new PopCommand(stack);
      popc1.execute();
      hlist.addCommand(popc1);
      System.out.println(stack);

      hlist.undo();
      System.out.println(stack);
  }
}
```

```
Stack [list=[6]]
Stack [list=[6, 3]]
Stack [list=[6]]
Stack [list=[6, 3]]
```

Command pattern

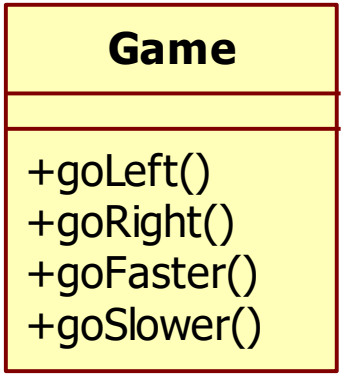
- What problem does it solve?
  - *Whenever you need to know the actions taken by a user, you can use the command pattern.*
  - *One important application of this pattern is undo/redo functionality.*
  - *Providing support for macros (recording and playback of macros).*

Issues

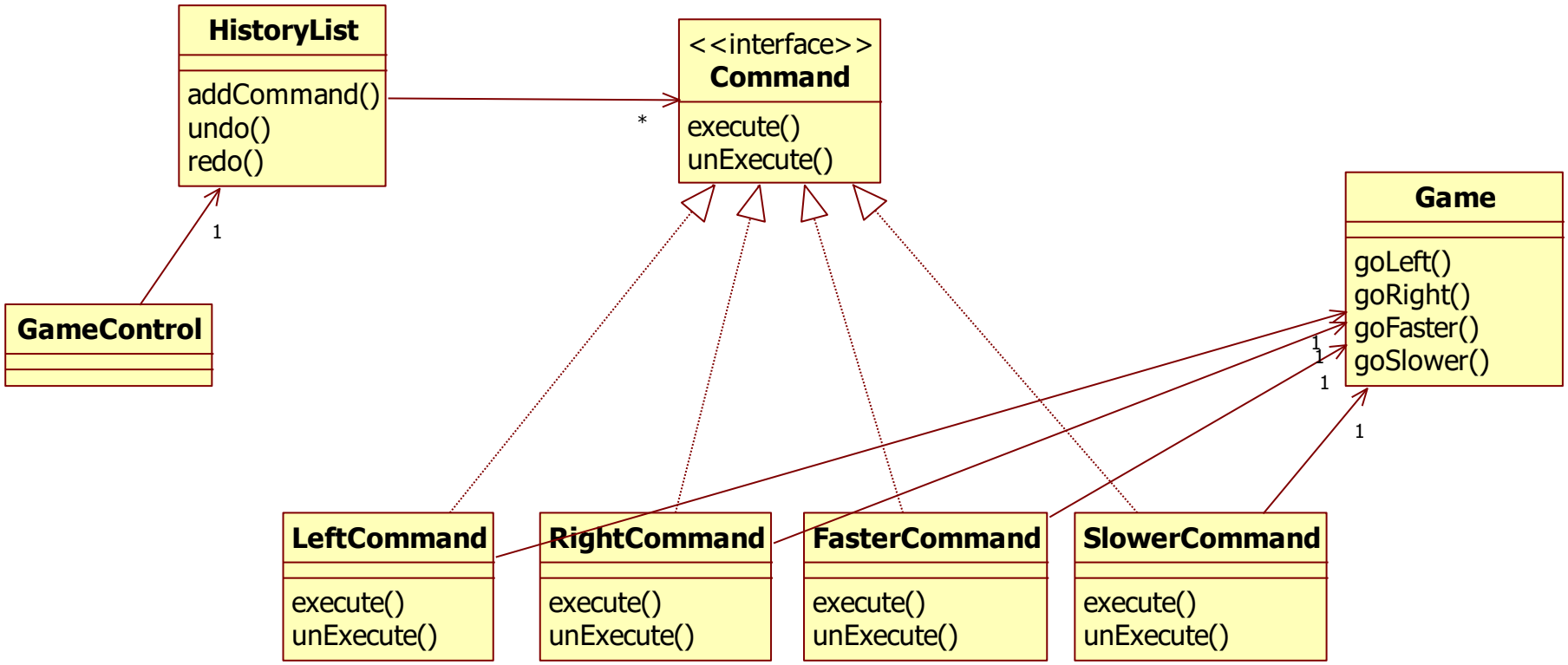
- How to store the state in the command object such that we can perform an undo by calling unExecute(). This state can be very complex.
- You can end up with a lot of command objects

Example

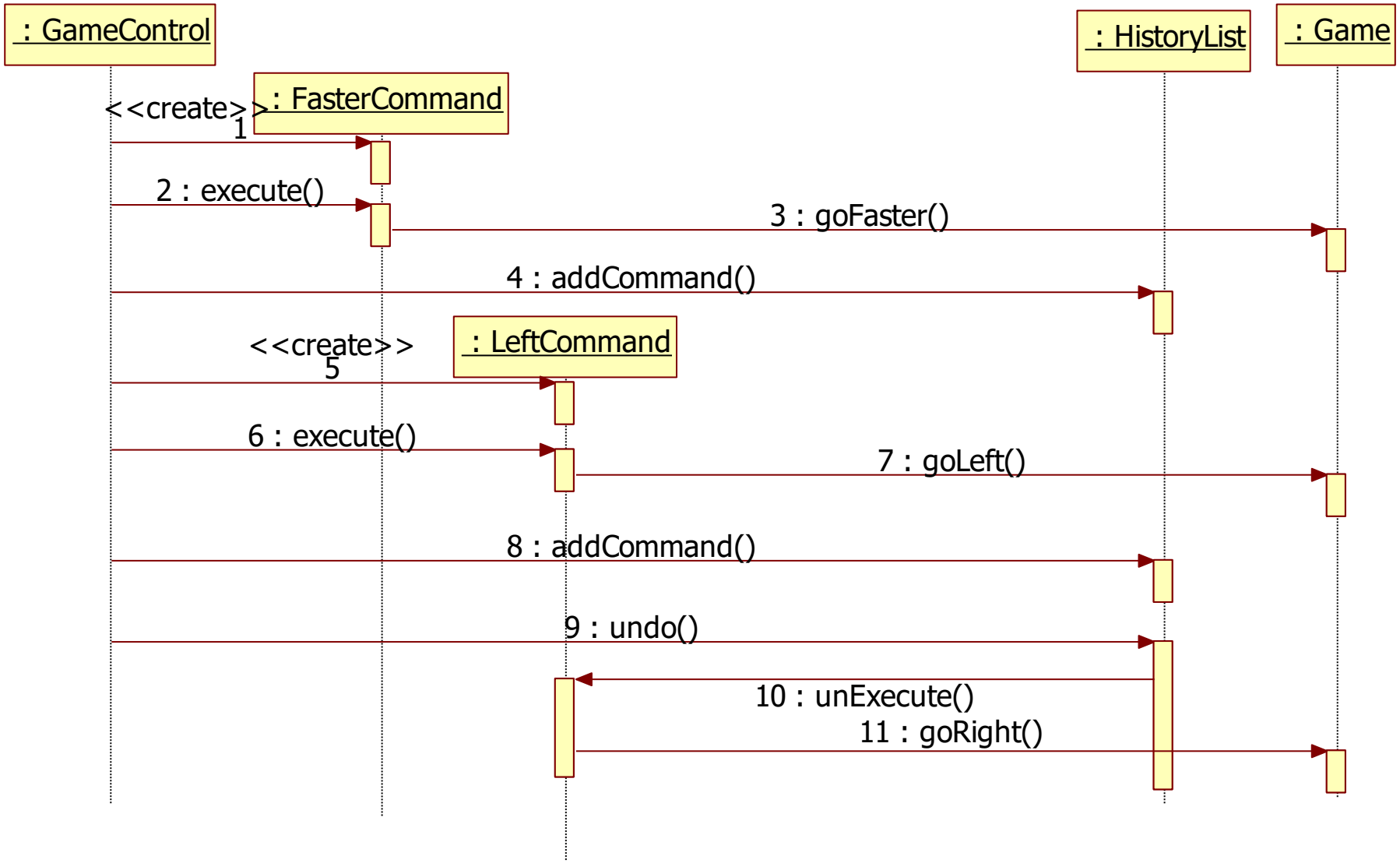
- Suppose we have a game that allows us to race with a car.
- The user can perform the following actions: go faster, go slower, go left and go right.
- We want to add functionality to record a whole race such that we can replay the recorded race.



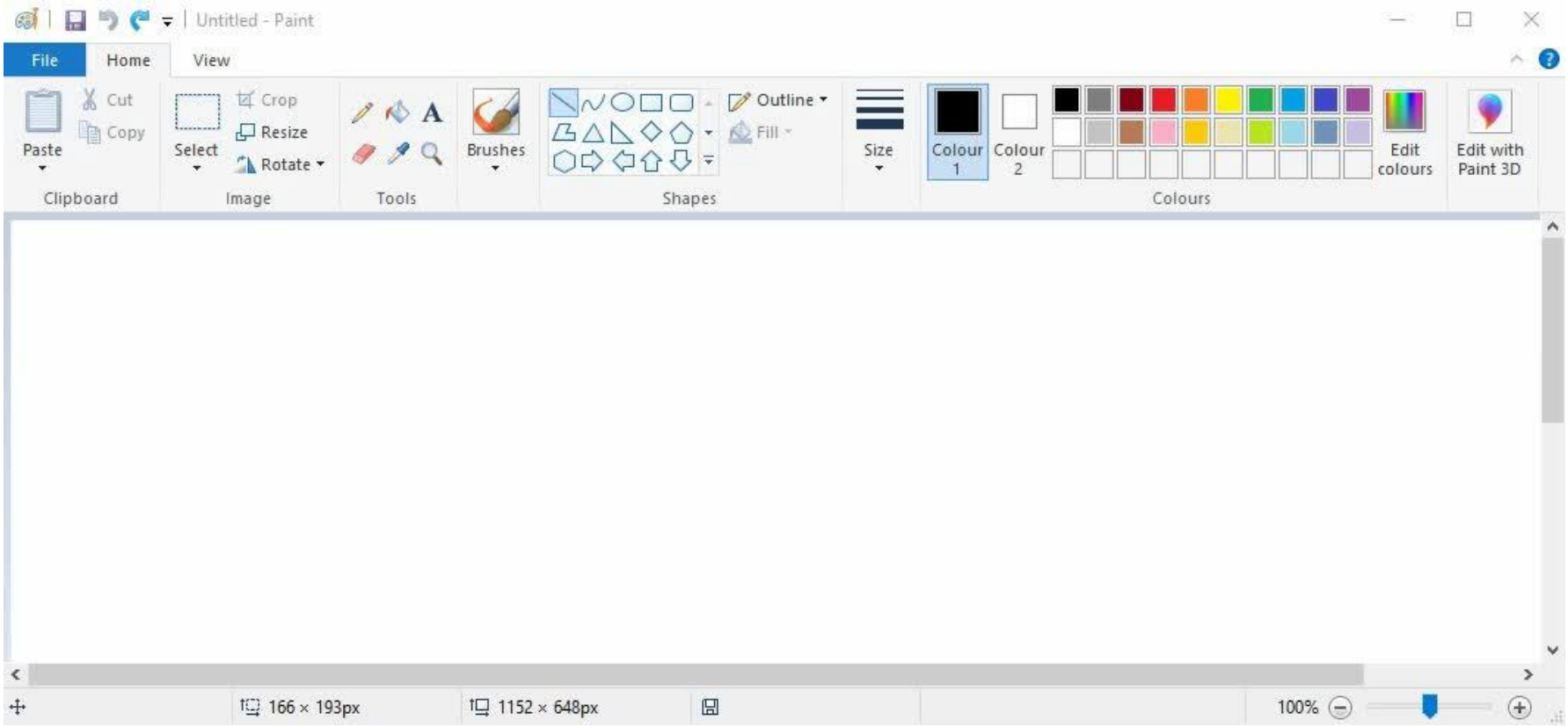
Example


HistoryList
addCommand()
undo()
redo()
<<interface>>
Command
execute()
unExecute()
*
1
GameControl
Game
goLeft()
goRight()
goFaster()
goSlower()
1
1
1
1
LeftCommand
execute()
unExecute()
RightCommand
execute()
unExecute()
FasterCommand
execute()
unExecute()
SlowerCommand
execute()
unExecute()

Example


: GameControl
: FasterCommand
: HistoryList
: Game
<<create>>
1
2 : execute()
3 : goFaster()
4 : addCommand()
<<create>>
5
: LeftCommand
6 : execute()
7 : goLeft()
8 : addCommand()
9 : undo()
10 : unExecute()
11 : goRight()

Example



Main point

- The Command pattern supports undo/redo functionality by storing state information in the Command objects.
- The Unified Field contains all knowledge in its most simple and abstract form

Connecting the parts of knowledge with the wholeness of knowledge

1. The command pattern encapsulates a request as an object.
2. Undo/redo functionality can be implemented by recording a HistoryList of Command objects.

3. **Transcendental consciousness** is the source off all activity.
4. **Wholeness moving within itself:** In Unity Consciousness, one experiences that you yourself (rishi), and all other objects (chhandas) and the interaction between yourself and all other objects (devata) are expressions of one's own Self.