Design Patterns

MSc in Communications Software

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Undo/Redo Command

Undo

- A single parameter-less command in the console - ‘undo’
- Undo last command
    - eg, if user just added, remove the user. If user removed, add back in

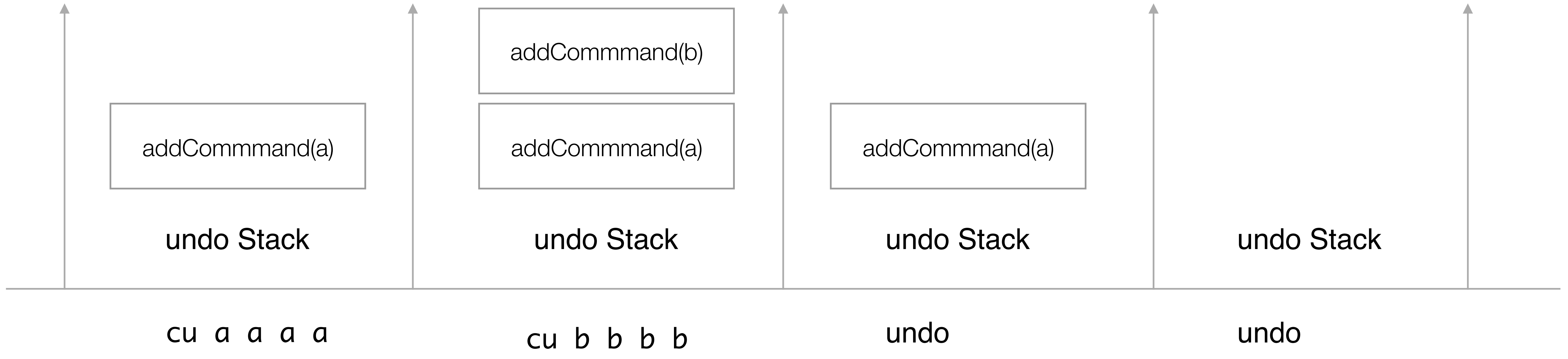
Undo Example

```
Welcome to pacemaker-console - ?help for instructions
pm> cu a a a a
+----+-----------+----------+-------+----------+
| ID | FIRSTNAME | LASTNAME | EMAIL | PASSWORD |
+----+-----------+----------+-------+----------+
|  1 |         a |        a |     a |        a |
+----+-----------+----------+-------+----------+
pm> undo
pm> lu

pm>
```

```
Welcome to pacemaker-console - ?help for instructions
pm> cu a a a a
+----+-----------+----------+-------+----------+
| ID | FIRSTNAME | LASTNAME | EMAIL | PASSWORD |
+----+-----------+----------+-------+----------+
|  1 |         a |        a |     a |        a |
+----+-----------+----------+-------+----------+
pm> du 1
pm> lu

pm>
pm> undo
pm> lu
+----+-----------+----------+-------+----------+
| ID | FIRSTNAME | LASTNAME | EMAIL | PASSWORD |
+----+-----------+----------+-------+----------+
|  1 |         a |        a |     a |        a |
+--+-----------+----------+-------+----------+
pm>
```

undo Stack

- When a command is executed - push the command onto an ‘undo’ stack
- When undo is to be executed - pop the ‘undo’ stack and call ‘undoCommand’



Command with Undo Support

```
public abstract class Command
{
  protected PacemakerAPI pacemaker;
  protected Parser       parser;

  public Command()
  {}

  public Command(PacemakerAPI pacemaker, Parser parser)
  {
    this.pacemaker = pacemaker;
    this.parser    = parser;
  }

  public abstract void doCommand(Object[] parameters)  throws Exception;

  public abstract void undoCommand() throws Exception;
}
```

CreateUser with Undo

```
public class CreateUserCommand extends Command
{
  User user;

  public CreateUserCommand(PacemakerAPI pacemaker, Parser parser)
  {
    super(pacemaker, parser);
  }

  public void doCommand(Object[] parameters) throws Exception
  {
    Long id = pacemaker.createUser((String)parameters[0], (String)parameters[1],
                                   (String)parameters[2], (String)parameters[3]);
    System.out.println(parser.renderUser(pacemaker.getUser(id)));
    this.user = pacemaker.getUser(id);
  }

  public void undoCommand() throws Exception
  {
    pacemaker.deleteUser(user.id);
  }
}
```

DeleteUser with Undo

```java
public class DeleteUserCommand extends Command
{
  private User user;

  public DeleteUserCommand(PacemakerAPI pacemaker, Parser parser)
  {
    super(pacemaker, parser);
  }

  public void doCommand(Object[] parameters) throws Exception
  {
    this.user = pacemaker.getUser((Long)parameters[0]);
    pacemaker.deleteUser((Long)parameters[0]);
  }

  public void undoCommand() throws Exception
  {
    pacemaker.createUser(user.firstname, user.lastname, user.email, user.password);
  }
}
```

ListUser with Undo?

```
public class ListUsersCommand extends Command
{
  public ListUsersCommand(PacemakerAPI pacemaker, Parser parser)
  {
    super(pacemaker, parser);
  }

  public void doCommand(Object[] parameters) throws Exception
  {
    System.out.println(parser.renderUsers(pacemaker.getUsers()));
  }

 public void undoCommand() throws Exception
 {
  //??
 }
}
```

- Undo/Redo doesn't make sense with ListUsers

Revised Command

- Make undoCommand and empty (non-abstract) method in base class

- Only override if appropriate

```
public abstract class Command
{
  protected PacemakerAPI pacemaker;
  protected Parser       parser;

  public Command()
  {}

  public Command(PacemakerAPI pacemaker, Parser parser)
  {
    this.pacemaker = pacemaker;
    this.parser    = parser;
  }

  public abstract void doCommand(Object[] parameters)  throws Exception;

  public void undoCommand() throws Exception
  {}
}
```

ListUser with Undo?

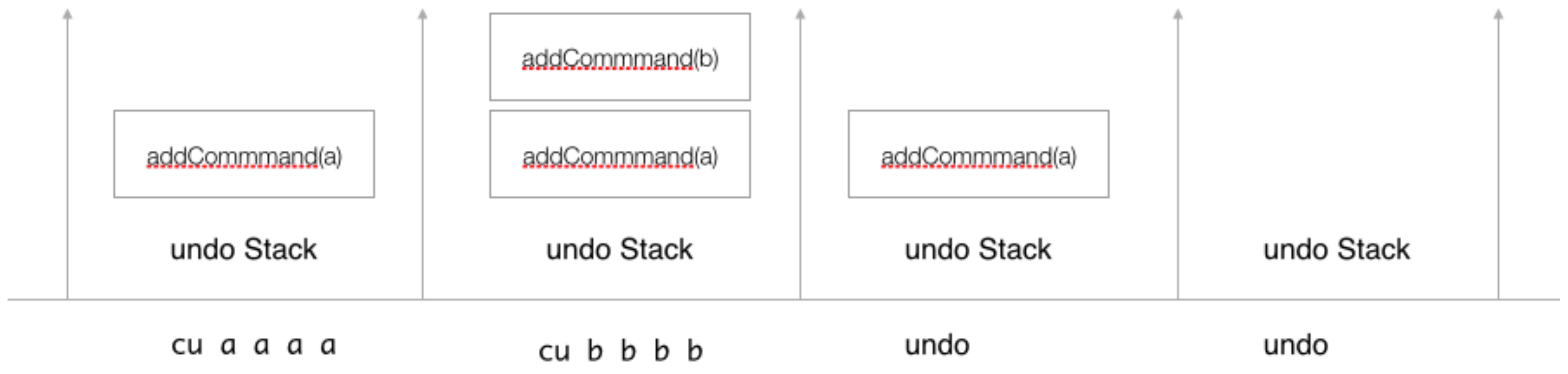
```
public class ListUsersCommand extends Command
{
  public ListUsersCommand(PacemakerAPI pacemaker, Parser parser)
  {
    super(pacemaker, parser);
  }

  public void doCommand(Object[] parameters) throws Exception
  {
    System.out.println(parser.renderUsers(pacemaker.getUsers()));
  }
}
```

- Undo/Redo doesn't make sense with ListUsers - so accept default implementation

How to implement 'undo'

- When a command is executed - push the command onto an 'undo' stack
- When undo is to be executed - pop the 'undo' stack and call 'undoCommand'

- Consider making undo a command in its own right
- i.e. encapsulate the 'undo' behaviour in a class derived from Command



Undo Command

- Undo is a command like any other

- when executed, it pops the ‘undo’ stack, invokes the ‘undoCommand’ method

```
public class UndoCommand extends Command
{
  private Stack<Command> undoBuffer;

  public UndoCommand(Stack<Command> undoBuffer)
  {
    this.undoBuffer = undoBuffer;
  }

  public void doCommand(Object[] parameters) throws Exception
  {
    if (undoBuffer.size() > 0)
    {
      Command command = undoBuffer.pop();
      command.undoCommand();
    }
  }
}
```

```
public class CommandDispatcher
{
  private Map<String, Command> commands;
  private Stack<Command>       undoBuffer;

  public CommandDispatcher()
  {
    undoBuffer = new Stack<Command>();
    commands   = new HashMap<String, Command>();

    commands.put("undo", new UndoCommand(undoBuffer));
  }

  public void addCommand(String commandName, Command command)
  {
    commands.put(commandName, command);
  }

  public boolean dispatchCommand(String commandName, Object [] parameters) throws Exception
  {
    boolean dispatched = false;
    Command command = commands.get(commandName);

    if (command != null)
    {
      dispatched = true;
      command.doCommand(parameters);
      undoBuffer.push(command);
    }
    return dispatched;
  }
}
```

- undoBuffer created by CommandDispatcher.
- It is passed to UndoCommand constructor.

CommandDispatcher

```
public class PacemakerShell implements CommandProcessor
{
  private CommandDispatcher dispatcher;
  private PacemakerAPI       paceApi;

  public PacemakerShell()
  {
    Parser parser = new AsciiParser();
    paceApi     = new PacemakerAPI();
    dispatcher = new CommandDispatcher();
    dispatcher.addCommand("list-users",  new ListUsersCommand(paceApi,  parser));
    dispatcher.addCommand("create-user", new CreateUserCommand(paceApi, parser));
    dispatcher.addCommand("delete-user", new DeleteUserCommand(paceApi, parser));
  }

  @Override
  public void doCommand(ShellCommand command, Object[] parameters)
  {
    try
    {
      dispatcher.dispatchCommand(command.getName(), parameters);
    }
    catch (Exception e)
    {
      System.out.println("Error executing command");
    }
  }

  public static void main(String[] args) throws Exception
  {
    PacemakerShell main = new PacemakerShell();
    CommandSpecifications commandSpecs = new CommandSpecifications();

    Shell shell = ShellFactory.createConsoleShell("pm", "Welcome to pacemaker-console - ?help for instructions",
                                                  commandSpecs, main);
    shell.commandLoop();
  }
}
```

- When user types undo - it is dispatched like any other command

CommandSpecifications

- Include ‘undo’ as parameterless command

```
public class CommandSpecifications
{
  @Command(description="List all users details")
  public void listUsers () throws Exception
  {}

  @Command(description="undo last command")
  public void undo () throws Exception
  {}

  @Command(description="Create a new User")
  public void createUser (@Param(name="first name") String firstname, @Param(name="last name") String lastname,
                          @Param(name="email")      String email,     @Param(name="password")  String password)
  {}

  @Command(description="Delete a User")
  public void deleteUser (@Param(name="id") Long id)
  {}
}
```

Redo

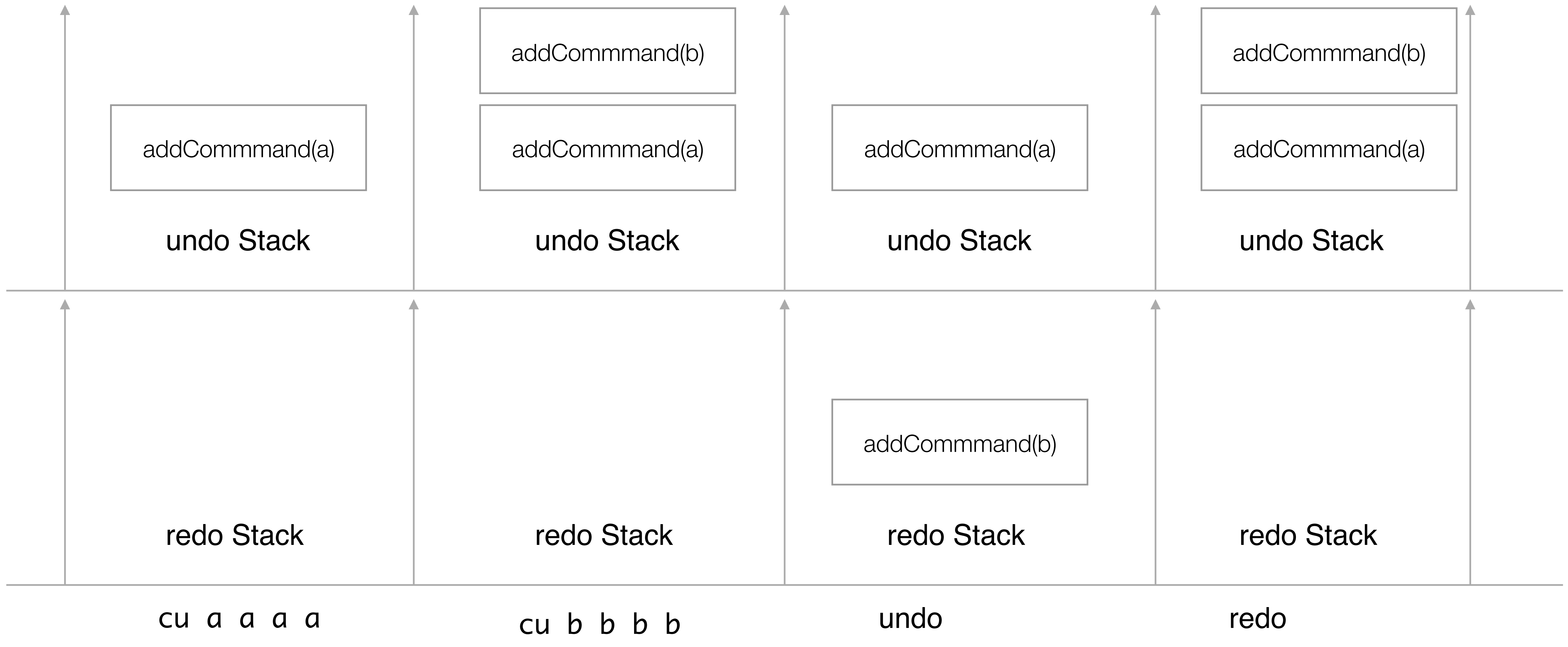
- Undo last command
    - eg, if user just added, remove the user. If user removed, add back in
- Redo last undo
    - eg, if user added, and undo implemented (to delete user), then redo should add user back in.

redo example

```
Welcome to pacemaker-console - ?help for instructions
pm> cu a a a a
+----+-----------+----------+-------+----------+
| ID | FIRSTNAME | LASTNAME | EMAIL | PASSWORD |
+----+-----------+----------+-------+----------+
|  1 |         a |        a |     a |        a |
+----+-----------+----------+-------+----------+
pm> cu b b b b
+----+-----------+----------+-------+----------+
| ID | FIRSTNAME | LASTNAME | EMAIL | PASSWORD |
+----+-----------+----------+-------+----------+
|  2 |         b |        b |     b |        b |
+----+-----------+----------+-------+----------+
pm> lu
+----+-----------+----------+-------+----------+
| ID | FIRSTNAME | LASTNAME | EMAIL | PASSWORD |
+----+-----------+----------+-------+----------+
|  1 |         a |        a |     a |        a |
|  2 |         b |        b |     b |        b |
+----+-----------+----------+-------+----------+
pm> undo
pm> lu
+----+-----------+----------+-------+----------+
| ID | FIRSTNAME | LASTNAME | EMAIL | PASSWORD |
+----+-----------+----------+-------+----------+
|  1 |         a |        a |     a |        a |
+----+-----------+----------+-------+----------+
pm> redo
pm> lu
+----+-----------+----------+-------+----------+
| ID | FIRSTNAME | LASTNAME | EMAIL | PASSWORD |
+----+-----------+----------+-------+----------+
|  1 |         a |        a |     a |        a |
|  3 |         b |        b |     b |        b |
+----+-----------+----------+-------+----------+
pm>
```

redo Stack

- When undo command is executed - pop the ‘undo’ stack, call ‘undoCommand’ and push command onto ‘redo’ stack.
- When redo is to be executed - pop the ‘redo’ stack, call ‘redoCommand’ and push onto a ‘undo’ stack…



redo operation

- Is redo just the same as doCommand?
- Not necessarily
    - doCommand may require user interaction
    - redoCommand will often not require any user interaction, and should use ‘remembered’ data to redo the command

Command with Undo and Redo Support

```
public abstract class Command
{
  protected PacemakerAPI pacemaker;
  protected Parser       parser;

  public Command()
  {}

  public Command(PacemakerAPI pacemaker, Parser parser)
  {
    this.pacemaker = pacemaker;
    this.parser    = parser;
  }

  public abstract void doCommand(Object[] parameters)  throws Exception;

  public void undoCommand() throws Exception
  {}

  public void redoCommand() throws Exception
  {}
}
```

CreateUser with Undo/Redo

```
public class CreateUserCommand extends Command
{
  User user;

  public CreateUserCommand(PacemakerAPI pacemaker, Parser parser)
  {
    super(pacemaker, parser);
  }

  public void doCommand(Object[] parameters) throws Exception
  {
    Long id = pacemaker.createUser((String)parameters[0], (String)parameters[1],
                                   (String)parameters[2], (String)parameters[3]);
    System.out.println(parser.renderUser(pacemaker.getUser(id)));
    this.user = pacemaker.getUser(id);
  }

  public void undoCommand() throws Exception
  {
    pacemaker.deleteUser(user.id);
  }

  public void redoCommand() throws Exception
  {
    pacemaker.createUser(user.firstname, user.lastname, user.email, user.password);
  }
}
```

DeleteUser with Undo/Redo

```java
public class DeleteUserCommand extends Command
{
  private User user;

  public DeleteUserCommand(PacemakerAPI pacemaker, Parser parser)
  {
    super(pacemaker, parser);
  }

  public void doCommand(Object[] parameters) throws Exception
  {
    this.user = pacemaker.getUser((Long)parameters[0]);
    pacemaker.deleteUser((Long)parameters[0]);
  }

  public void undoCommand() throws Exception
  {
    pacemaker.createUser(user.firstname, user.lastname, user.email, user.password);
  }

  public void redoCommand() throws Exception
  {
    pacemaker.deleteUser(user.id);
  }
}
```

Redo Command

- Mirror image of undo command:
  - pop and ‘redo’ command in redo stack
  - push command back onto undo stack

```
public class RedoCommand extends Command
{
  private Stack<Command> undoBuffer;
  private Stack<Command> redoBuffer;

  public RedoCommand(Stack<Command> undoBuffer, Stack<Command> redoBuffer)
  {
    this.undoBuffer = undoBuffer;
    this.redoBuffer = redoBuffer;
  }

  public void doCommand(Object[] parameters) throws Exception
  {
    if (redoBuffer.size() > 0)
    {
      Command command = redoBuffer.pop();
      command.redoCommand();
      undoBuffer.push(command);
    }
  }
}
```

- Undo command must be extended to support redo stack

redo & undo

```java
public class UndoCommand extends Command
{
  private Stack<Command> undoBuffer;
  private Stack<Command> redoBuffer;

  public UndoCommand(Stack<Command> undoBuffer, Stack<Command> redoBuffer)
  {
    this.undoBuffer = undoBuffer;
    this.redoBuffer = redoBuffer;
  }

  public void doCommand(Object[] parameters) throws Exception
  {
    if (undoBuffer.size() > 0)
    {
      Command command = undoBuffer.pop();
      command.undoCommand();
      redoBuffer.push(command);
    }
  }
}
```

```java
public class RedoCommand extends Com
{
  private Stack<Command> undoBuffer;
  private Stack<Command> redoBuffer;

  public RedoCommand(Stack<Command> undoBuffer, Stack<Command> redoBuffer)
  {
    this.undoBuffer = undoBuffer;
    this.redoBuffer = redoBuffer;
  }

  public void doCommand(Object[] parameters) throws Exception
  {
    if (redoBuffer.size() > 0)
    {
      Command command = redoBuffer.pop();
      command.redoCommand();
      undoBuffer.push(command);
    }
  }
}
```

CommandDispatcher

```
public class CommandDispatcher
{
  private Map<String, Command> commands;
  private Stack<Command> undoBuffer;
  private Stack<Command> redoBuffer;

  public CommandDispatcher()
  {
    undoBuffer = new Stack<Command>();
    redoBuffer = new Stack<Command>();
    commands   = new HashMap<String, Command>();

    commands.put("undo", new UndoCommand(undoBuffer, redoBuffer));
    commands.put("redo", new RedoCommand(undoBuffer, redoBuffer));
  }

  public void addCommand(String commandName, Command command)
  {
    commands.put(commandName, command);
  }

  public boolean dispatchCommand(String commandName, Object [] parameters) throws Exception
  {
    boolean dispatched = false;
    Command command = commands.get(commandName);

    if (command != null)
    {
      dispatched = true;
      command.doCommand(parameters);
      undoBuffer.push(command);
    }
    return dispatched;
  }
}
```

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit