

Developing shells with jpf-shell

What exactly is a shell and what does it do and not do?

A shell really boils down to being nothing more than a swing app that can be launched from within jpf. Inside of the `JPF.main(String[] args)` function after a `Config` object is created and fully evaluated JPF checks if the "shell" property is set. If the "shell" property is set and points to a class that implements the `JPFShell` interface then an instance of it is constructed through a constructor that takes only a `Config` object. (This is the same config that jpf already created). Then the `"start(String[] args)"` method (the only method defined in `JPFShell` interface) is run and `JPF.main(String[] args)` is done leaving everything in the hands of the shell.

Where jpf-shell comes in

jpf-shell serves as a starting point for developing your shell. The handful of classes found in the "gov.nasa.jpf.shell" package make up a running instance of jpf-shell.

Here are some fundamental design ideas to keep in mind, all of these classes are meant to be subclassed:

gov.nasa.jpf.shell.ShellCommand

- Actually *does* work with the `execute()` method. (ie: Running JPF, Running the System Under test)
- Listeners can be attached to any `ShellCommand` to execute *before* and *after* they execute()
- `ShellCommands` are added to the `ShellManager`
- `ShellCommands` are fired through `ShellManager.fireCommand()`

gov.nasa.jpf.shell.ShellPanel

- extends `JPanel`
- Displayed by subclasses of `gov.nasa.jpf.shell.Shell`

gov.nasa.jpf.shell.Shell

- Extends `JFrame` and is displayed to the user
- Holds a group of panels
- Is responsible to display `ShellCommands` to the user

gov.nasa.jpf.shell.ShellManager

- Holds the following:
 - All `ShellCommands`
 - All `ShellCommandListeners`
 - All `Shells`
 - The `Config` object
- A Singleton class, only 1 exists and only one 1 is set for the lifetime of the JVM
- Responsible for I/O, error logging and communicating with IDEs.

Customizing

How do I make a new ShellCommand?

This example will create a new "HelloWorld" command and use the `BasicShell` as the `Shell` implementation. The culmination of this example can be found in the 'example/commands' folder in the jpf-core project.

1. Create a java class that extends `ShellCommand` like the following:

```
public class HelloWorldCommand extends ShellCommand{
    @Override
    public void execute() {
        // Do whatever you want here
        JOptionPane.showMessageDialog(null, "Hello, world!", "Hello from jpf-shell",
            JOptionPane.INFORMATION_MESSAGE);
    }
}
```

1. Extend `BasicShell` to create a new shell that will add our custom command to the `ShellManager`

```
public class HelloWorldCommandShell extends BasicShell{

    public HelloWorldCommandShell(Config c){
        super(c);//The Shell Constructor that takes a Config as an argument makes sure
            //to create a ShellManager for us.
    }

    @Override
    protected void addCommands(){
        super.addCommands(); //Add on all of the commands that BasicShell wants
        ShellManager.getManager().addCommand(new HelloWorldCommand());
    }
}
```

1. Add our new shell as the 'shell' property in our configuration.

```
shell=commands.HelloWorldCommandShell
```