Hayri Eryürek

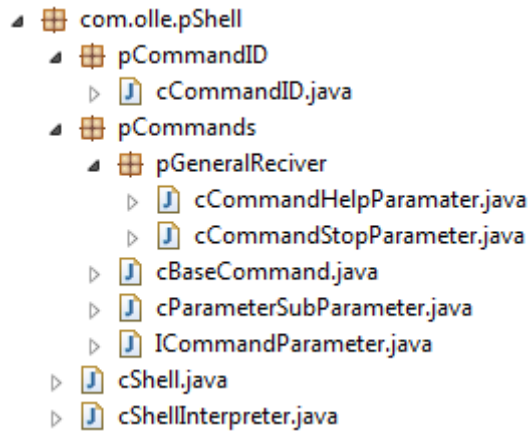
CommandShell.jar Kütüphanesi

İçindekiler

CommandShell.jar Kütüphanesi

CommandShell.jar kütüphanesi java'da yazılmış uygulamalarınıza komut arayüzü geliştirebilmeniz amacıyla Hayri Eryürek tarafından geliştirilmiştir. Generic yapısı sayesinde istediğiniz kadar komut ve istedeiğiniz komuta istediğiniz kadar parametre geliştirebilme imkanı sağlar. Konsolun Thread yapısı gereği her komut için bir Thread açılarak heran konsolun hizmet vermesi sağlanır. Kütüphane içerisindeki obje ve paket yapısı aşağıdaki gibidir.

Kütüphaneyi Oluşturan Sınıfların Listesi



CommandShell.jar Kütüphanesinin Dışarıya Açık Sınıflarının Metod ve Değiken İçerikleri :

cCommandID.java

```
package com.sanalstil.pShell.pCommandID;
/**
 * Komut oluşturma aşamasında cBaseCommand objesinin içine verilmek üzere tasarlanmıştır.
 * Bu obje komutun konsoldan çağrılma adını ve ID'sini içerir
 * @author Hayri Eryürek
 * @version Tarih : 07.04.2011 - Version 1.0
 */
public class cCommandID
{
        /**
         * Komut ID'si.
         */
        public int CommandID;
        /**
         * Komut Adı.
         */
        public String CommandName;

        /**
         * Komutun konsoldan çağrılma adını ve ID'si girmeliziniz.
         * @param _CommandID tipi Integer. Komut ID'si.
         * @param _CommandName tipi String. Komut Adı.
         */
        public cCommandID(int _CommandID, String _CommandName);
}
```

cParameterSubParameter.java

```
package com.sanalstil.pShell.pCommands;

import java.util.List;

/**
 * Komut parametresi ve o parametreye ait alt parametrelerin listesini içerir.
 * @author Hayri Eryürek
 * @version Tarih : 07.04.2011 - Version 1.0
 * @see cBaseCommand
 */
public class cParameterSubParameter
{
	/**
	 * Parametre Adı. örnek "-stop"
	 */
	public String Parameter;
	/**
	 * Parametre alt parametre listesi. (Örnek : list -p page1 page2) Yandaki örnekte page1
ve page2 alt parametrelerdir.
	 * Bu örnek için liste içinde -p parametresinin alt parametresi olan "page1" ve "page2"
kelimeleri bulunmaktadır.
	 */
	public List<String> SubParameters;

	/**
	 * @param _Parametre tipi String. Ana parametreyi içerir. örnek : "-p"
	 * @param _SubParameter String tipi bir liste içerir. Ana parametrenin Alt
parametrelerini içerir. örnek : {"page1", "page2"}
	 */
	public cParameterSubParameter(String _Parameter, List<String> _SubParameter);
}
```

ICommandParameter.java

```
package com.sanalstil.pShell.pCommands;

import java.util.List;

/**
 * Komutlara parametre alıcısı olarak eklenmek istenen objelerin komutlara bağlanabilmesi için
gliştirilmiş bir arayüzedür.
 * @author Hayri Eryürek
 * @version Tarih : 07.04.2011 - Version 1.0
 * @see cBaseCommand
 */
public interface ICommandParameter
{
	/**
	 *
	 * @param _Command parametresi cBaseCommand obje tipindendir. Parametreye objesine
hangi komuttan gelindiğini
	 * bildirmek amacı ile kullanılır.
	 * @param _SubParameter parametresi String tipinden Liste içerir. Parametrenin alt
parametrelerine erişmek amacı ile kullanılır.
	 */
	void ReciveCommand(cBaseCommand _Command, List<String> _SubParameter);
	/**
	 *
	 * @return Konsoldan parametre yönlendirmesi yapılabilmesi amacı ile kullanılır. örnek
: return "-help";
	 */
	String GetCallParameter();
	/**
	 *
	 * @return Kullanıcıya bu parametrenin amacını yardım istediğinde gösterebilmek amacı
ile kullanılır.
	 * örnek "return "Yardım Komutu";
	 */
	String GetHelpString();
}
```

cBaseCommand.java

```
package com.sanalstil.pShell.pCommands;

import java.util.LinkedList;
import java.util.List;

import com.sanalstil.pShell.cShell;
import com.sanalstil.pShell.pCommandID.cCommandID;
import com.sanalstil.pShell.pCommands.pGeneralReciver.cCommandHelpParamater;
import com.sanalstil.pShell.pCommands.pGeneralReciver.cCommandStopParameter;


/**
 * Shell komut ara yüzüne, yeni komut eklemek istediğinizde bu objeden türetmeniz
gerekmektedir.
 * Türettiğiniz bu komut içinde abstract InterpretCommand metodunu ve abstract IsMultiRunnable
metodunu override etmeniz gerekmektedir.
 * Türettiğiniz bu komuttan tek bir instance oluşturmalısınız daha fazlasını değil.
Oluşturulan komut instance'ı Shell içrisindeki komut listesine kendini ekleyecektir.
 * Artık türetilen bu komut konsol aracılığı ile çağırıldığında  override edilen
InterpretCommand metodu çağrılarak türettiğiniz yeni komuta devredilecektir.
 * InterpretCommand metodu List(String) tipinden _CommandParameter parametresi ve
List(cParameterSubParameter) tipinden _ParameterList parametresi olmak üzere iki parametre
almaktadır.
 * _CommandParameter parametresi "-" ile başlamayan ve hemen komuttan sonra kullanılmış olan
alt parametre listesini içerir.
 * _ParameterList parametresi ise cParameterSubParameter tipinden objeler listesi içerir. Bu
objelerin her birinin içinde iki değişken vardır.
 * Birisi Parameter adında string tipinde "-" ile başlamış parametrenin kendisidir.
 * İkincisi ise parametreden hemen sonra kullanılmış "-" başlamayan alt parametre listesini
içeren SubParameters adında String tipi içeren listedir.
 * Bu method içinden parametrelere devredilebilir, parametrelerle komutu init edebilir yada
direk olarak bu method içinden işleminizi gerçekleştirebilirsiniz.
 * Komutun her çağrıldığında komut için bir Thread açılarak çalıştırılır. Bu sayede konsol
meşgul edilmemiş olur.
 * Fakat bazen komut tamamlanmadan aynı komut tekrar çağrılmasını engelemek istiyebilirsiniz.
Bunun içinde, oluşturulan yeni komutta
 * override etmek zorunda olduğunuz IsMultiRunnable fonksiyonu devreye girmektedir. Eğer bu
komut çoklu çalışabiliyorsa
 * IsMultiRunnable fonksiyonu true, çalışamıyorsa false döndürmelidir.
 *
 * @author Hayri Eryürek
 * @version Tarih : 07.04.2011 - Version 1.0
 * @see cShell, cCommandID, ICommandParameter, cParameterSubParameter
 */

public abstract class cBaseCommand implements Runnable
{
	/** Komutun bağlı olduğu Shell */
	public cShell OwnerShell = null;
	/**Komut Adını ve ID'sini içeren cCommandID objesi */
	public cCommandID CommandID = null;
	/** Kendini komutun alıcısı olarak ekleyen ICommandParameter arayüzünden türemiş
objelerin listesi */
	public List<ICommandParameter> CommandParameter = new LinkedList<ICommandParameter>();
	/** Komutun son kullanıldığındaki parametre ve bu parametrelerinin alt parametre
listelerini tutan cParameterSubParameter objesinin listesi /
	public List<cParameterSubParameter> ParameterList = null;
	/** Komutun son kullanıldığındaki alt parametre listesi */
	public List<String> CommandSubParameter = null;
/**	 * Komutun her kullanıldığında açılan Thread'lerin listesi
	 * Bütün Thread'ler sonlanana dek liste temizlenmez. Yani gerçekte son bulmuş
Thread'ler varsabile tüm Thread'ler
	 * son bulunca liste temizlenecektir. */
	public List<Thread> OwnerThreadList = null;
/**	 * @param _OwnerShell Parametresi komutun bağlanacağı Shell'i içermelidir.
	 * @param _CommandID Parametresi tipi cCommandID obje tipindendir. cCommandID objesi
içinde CommandName değişkeninde komutun çağrılacağı ad,
	 * CommandID içinde ise bu komuta ait ID numarası yer almalıdır.
	 */
	public cBaseCommand(cShell _OwnerShell, cCommandID _CommandID);
	/** Komuta ait parametre listesini yardım olarak listeler. */
	public void PrintHelp();
```

```
/**
         * Komuta bağlanmış olan parametre listesi içinden parametre kelimesini verip (örnek :
"-s" ) parametre objesini
         * almak için kullanılır.
         * @param _Parameter tipi String olup parametre kelimesi içerir
         * @return Donen obje ICommandParameter tipindendir. Aranan kelimeye ait bir obje
listede bulunuyorsa döner. Bulunmuyorsa null döner.
         */
        public ICommandParameter GetParameterClassByParameter(String _Parameter);
        /**
         * Kendini ilgili komutun alıcı olarak eklemek isteyen ICommandParameter arayüzünden
türemiş
         * objelerin komuta bağlanması için kullanılır.
         * @param _CommandReciver tipi ICommandParameter olup komut alıcısını içerir.
         * @return Donen deger boolean tipinden olacaktır. Eğer aynı parametre adına sahip
başka bir parametre
         * objesi yoksa bağlantı sağlanacak ve True değeri dönecektir. Eğer başka bir parametre
objeside bu komuta
         * aynı isimle bağlnmışsa aynı isimle ikinci bir parametre daha bağlanamayacağı için
False değeri dönecektir.
         */
        public boolean Connect(ICommandParameter _CommandReciver);
        /**
         * Kendini ilgili komutun alıcı olarak eklemek isteyen ICommandParameter arayüzünden
türemiş
         * objelerin komuta bağlanması için kullanılır.
         * @param _CommandReciver tipi ICommandParameter olup komut alıcısını içerir.
         * @return Donen değer boolean tipinden olacaktır. Bağlantısı kesilmek istenen obje
hali hazırda bu komuta bağlıysa
         * bağlantısı
         */
        public boolean Disconnect(ICommandParameter _CommandReciver);
        /**
         * Konsole aracılığı ile, yeni türettiğiniz komut çağrıldığında, override edilen
InterpretCommand metodu sizin komutunuzdan çağrılacaktır.
         */
        public abstract void InterpretCommand(List<String> _CommandParameter,
List<cParameterSubParameter> _ParameterList);

        /**
         * Türettiğiniz komutun aynı anda birden fazla şekilde çalışabilmesini istiyorsanız
override ettiğiniz IsMultiRunnable
         * metodu True istemiyorsanız False döndürmelidir.
         */
        public abstract boolean IsMultiRunnable();
        /** Komutunuz konsola birşey basmak istediğinde kullanılır. */
         public void PrintConsole(Object _Object);
/**      * Komutunuzun o anda çalıp çalışmadığını kontrol etmek amacıyla kullanılır.
         * Aktif bir komut thread'i varsa True yoksa False dönecektir.
         */
         public boolean IsRunning()
        /** Komutunuzun o anda çalır durumdaki tüm Threadlerini kapatmak için kullanılır.*/
         public void Stop()

}
```

cShell.java

```
package com.sanalstil.pShell;

import java.util.LinkedList;
import java.util.List;

import com.sanalstil.pShell.pCommands.cBaseCommand;

/**
 * Uygulamalarınıza, Konsol arayüzü olmak amacıyla tasarlanmıştır.
 * @author Hayri Eryürek
 * @version Tarih : 07.04.2011 - Version 1.0
 * @see Shell
 */
public class cShell
{
	/** Shell'de kullanılabilecek komut listesini içerir. */
	public List<cBaseCommand> CommandList = null;

	public cShell();

  /**
   * cBaseCommand Objesinden türetilen her objenin instance'ı, liste içine eklenir. Bu komut sayesinde parametre olarak verilen komut ismi,
   * liste içindeki komutların oluşturulma aşamasında verilen isimleri ile karşılaştırılarak aranır.
   * Eğer bu isimde bir komutu bulursa türediği cBaseCommand obje cinsinden obje olarak döner.
   * Bulamazsa sonuç null olarak döner.
   * @param _CommandName parametresi, String tipinde bir parametredir.
   * Bu metod, _CommandName ile komutlar oluşturulurken cCommandID içinde verilen CommandName değişkenlerinde karşılaştırma yapar.
   *
   * @return Bu metodun sonucu olarak cBaseCommand obje cinsinden bir komut objesi, Komut bulunamazsa null sonuç döner.
   */
	public cBaseCommand GetCommandByName(String _CommandName);
	/**
	   * cBaseCommand Objesinden türetilen her obje, liste içine eklenir. Bu komut sayesinde parametre olarak verilen komut ID si,
	   * liste içindeki komutların oluşturulma aşamasında verilen ID leri ile karşılaştırılarak aranır.
	   * Eğer bu ID de bir komutu bulursa türediği cBaseCommand obje cinsinden obje olarak döndürür.
	   * Bulamazsa sonuç null olarak döner.
	   * @param _CommandID parametresi, int tipinde bir parametredir.
	   * Bu metod, _CommandID ile komutlar oluşturulurken cCommandID içinde verilen CommandID değişkenlerinde karşılaştırma yapar.
	   *
	   * @return Bu metodun sonucu olarak cBaseCommand obje cinsinden bir komut objesi döner. Eğer komut bulunamazsa null sonuç döner.
	   */
	public cBaseCommand GetCommandByID(int _CommandID);
	 /** Shell oluşturulduktan sonra komutlar ve parametreler eklenir.
	   * Daha sonra Shell'i başlatmak için StartShell() komutu çağrılır.
	 **/
	public void StartShell();

	/** Konsola birşeyler yazdırabilmek için PrintConsole() komutu kullanılır.
	  * @param _Object parametresi Object tipinden her objeyi alablir.
	 **/
	public void PrintConsole(Object _Object);

	/** Shell'i kapatmak için StopShell()komutu çağrılır. */
	public void ExitShell();
}
```

Kütüphane Kullanım Örneği

cTestCommand.java

Yeni bir komut oluşturabilmek için öncelikle cBaseCommand sınıfından türetmemiz gerekiyor. Biz aşağıdaki örnekte cTestCommand adı altında bir objeyi cBaseCommand objesinden türettik. Komutumuz hangi shell sınıfına bağlanacağını bilmesi için cShell tipinden bir objeyi parametre olarak alıyor. cBaseCommand objesine ise super komutu aracılığı ile bağlanacağı shelli gönderiyor ve ikinci parametre olarakta komutun konsoldan çağrılacağı adı ve komutun ID'sini verebilmek amacıyla cCommandID objesinden oluşturarak cBaseObject sınıfına gönderiyoruz. Biz örneğimizde "Test" kelimesi konsoldan çağrıldığında bu komuta yönlenmesini istedik. cBaseCommand objesinden gelen override etmemiz gereken 2 metodumuz mevcut. Bunlardan biri **InterpretCommand**, diğeri **IsMultiRunnable** metodlarıdır. InterpretCommand metodu konsoldan bizim komutumuz çağrıldığında, çağrılacak metotdur. IsMultiRunnable metodu ise bizim komutumuzun çoklu çalışıp çalışamayacağını CommandShell kütüpanesine bildirmek amacıyla override etmek zorunda olduğumuz bir metotdur.

```
package pTest;

import java.util.LinkedList;
import java.util.List;

import com.sanalstil.pShell.cShell;
import com.sanalstil.pShell.pCommandID.cCommandID;
import com.sanalstil.pShell.pCommands.cBaseCommand;
import com.sanalstil.pShell.pCommands.cParameterSubParameter;

public class cTestCommand extends cBaseCommand
{
	public cTestCommand(cShell _OwnerShell)
	{
		super(_OwnerShell, new cCommandID(0, "Test"));
	}

	@Override
	public void InterpretCommand(List<String> _CommandParameter,
	List<cParameterSubParameter> _ParameterList)
	{
		NonParameterRun();
		for (int i = 0; i < _ParameterList.size();i++)
		{
			cParameterSubParameter __Parameter = _ParameterList.get(i);
			GetParameterClassByParameter(__Parameter.Parameter).ReciveCommand(this,
			__Parameter.SubParameters);
		}
	}

	private void NonParameterRun()
	{
		for (int i = 0;i < CommandParameter.size(); i++)
		{
			if (CommandParameter.get(i).GetCallParameter().equals(""))
			{
				CommandParameter.get(i).ReciveCommand(this,
				new LinkedList<String>());
			}
		}
	}

	@Override
	public boolean IsMultiRunnable()
	{
		return true;
	}
}
```

Konsola “Test C:\Deneme -t 500 -k Alt1 Alt2 Alt3” yazılıp test komutu çağrıldığında, bizim komutumuzun override ettiğimiz InterpretCommand metodu çağrılır. Metodun ilk parametresi olan _CommandParameter parametresi String tipinden bir liste içerir. Bu listenin içeriği parametresiz yazılmış direk komutun alt parametre listesidir. Çalıştırdığımız komut için bu listesinin içinde “C:\Deneme” tek eleman olarak gelcek. InterpretCommand metodunun ikinci parametresi olan _ParameterList parametresi cParameterSubParameter obje tipinden içeriği olan bir liste içerir. cParameterSubParameter objesinin içieri ise şöyledir.

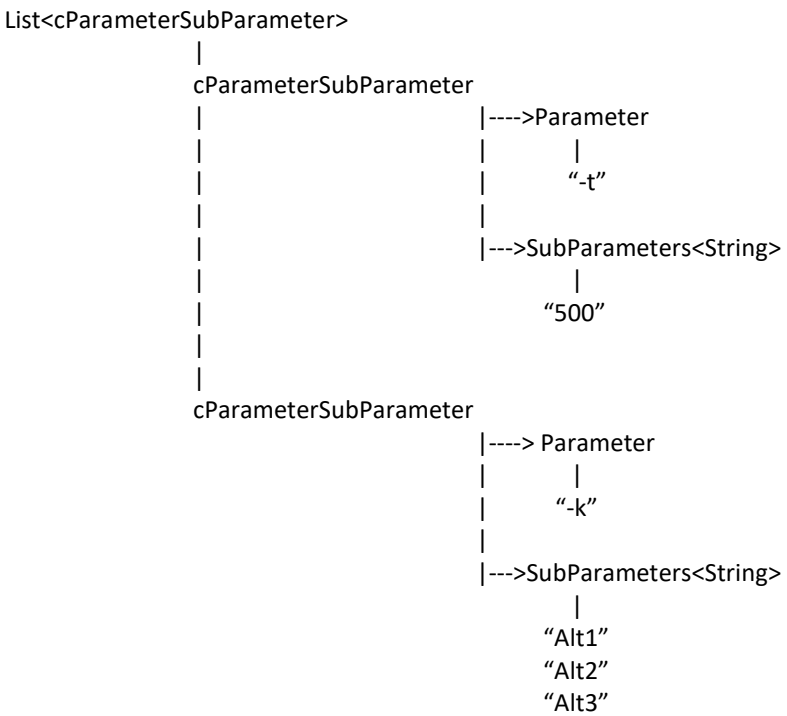
cParameterSubParameter.java

```
package com.sanalstil.pShell.pCommands;

import java.util.List;

public class cParameterSubParameter
{
	public String Parameter;
	public List<String> SubParameters;
	public cParameterSubParameter(String _Parameter, List<String> _SubParameter)
	{
		Parameter = _Parameter;
		SubParameters = _SubParameter;
	}
}
```

Yani InterpretCommand metodunun 2 parametresi olan _ParameterList parametresi cParameterSubParameter objesinin listesini tutar her bir objenin içinde ise parametre ve alt parametre listesi tutulur. Yukarıdaki Shell komut satırına yazılan örnek için bize _ParameterList ile gelecek parametre içeriği şöyledir.



Buna göre yukarıdaki test komutunun InterpretCommand metoduna devredilir. Yukarıdaki örnekte, işlemi devralan komut öncelikle koşulsuz kendini komuta bağlayan parametre objelerine devretmiş. Koşulsuz parametrelerin işlemi bittikten sonra konsolumuza girdiğimiz “-t“ parametresini kendi içindeki parametre listesinden talep eder ve o parametreye devreder. O parametre işilemini bitridikten sonra, sıradaki parametre olan “-k“ parametresini, kendi içindeki parametre listesinden talep eder ve o parametre objesine işlemlerini gerçekleştirmesi için devreder.

cCommandID.java

cCommandID objesi, cBaseCommand objesinden türetilmiş Shell komutlarının, Konsol çağrılma isimlerini ve ID'lerini cBaseCommand temel objesine bildirmek için kullanılmaktadır. Hiç bir özel metod içermeyen sadece structure gibi kullanılan bir objedir.

```
package com.sanalstil.pShell.pCommandID;

public class cCommandID
{
	public int CommandID;
	public String CommandName;

	public cCommandID(int _CommandID, String _CommandName)
	{
		CommandID = _CommandID;
		CommandName = _CommandName;
	}
}
```

cTestCommandNonParameterRun.java

Her hangi bir komut için parametre geliştirmek isteniyorsa ICommandParameter arayüzü implament edilmelidir. Çünkü komutun Connect metodu ICommandParameter arayüz tipinden bir obje almaktadır. Bu arayüzle gelen 3 tane metodumuz implament edilmiş olur. Bunlardan birisi GetCallParameter metodudur. Bu metoda konsolda ne yazılınca çağrılması gerektiğini belirtmeliyiz. Örnek olarak "-t" olabilir. Eğer her koşulda çalışmasını istiyorsak boş String döndürmemiz uygun olacaktır. Tabi bu sizin komutunuzun tasarımına göre değişiklik gösterebilir. Bir diğer metodumuz ise GetHelpString metodudur. Bu metod ise geliştirdiğiniz parametrenin yardım String'i dir. Her komuta otomatik olarak "-stop" ve "-help" parametreleri eklenir. Hatalı parametre girilmesi durumunda yada "-help" parametresi kullanılması durumunda vermiş olduğunuz yardım String'i parametrenizi açıklayacaktır. Son metodumuz olan ReciveCommand metodumuz ise komutumuzla birlikte parametremiz kullanıldığında parametre objemize yapması gereken işlemleri yapması için çağrılacaktır. Bu metod 2 paratre alır. Birisi hangi komutun bu parametreye yönlendirdiğini gösteren cBaseCommand temel tipinden olan _Command parametresidir. Diğeri ise parametre ile verilmiş alt parametrelerin String tipinde listesini içeren _SubParameter parametresidir.

```
package pTest;

import java.util.List;

import com.sanalstil.pShell.pCommands.ICommandParameter;
import com.sanalstil.pShell.pCommands.cBaseCommand;

public class cTestCommandNonParameterRun implements ICommandParameter
{


	public String GetCallParameter()
	{
		return "";
	}

	public String GetHelpString()
	{
		return null;
	}

	public void ReciveCommand(cBaseCommand _Command, List<String> _SubParameter)
	{
		_Command.PrintConsole("Komut parametresiz çağrıldığındada çalışır");
	}

}
```

cTestCommandParameter_b.java

cTestCommandNonParameterRun parametre objesi için anlattıklarımız bu parametre objesi içinde geçerlidir. cTestCommandNonParameterRun objesinden farkı bir parametre ile çağrıllabilmesidir. Aşağıda görüldüğü üzre komutumuz kullanılırken "-b" parametresi ile çağrıldığında bu prametre objesine de devredilecek. Devredilecek metod daha önceden de belirtiğimiz gibi ReciveCommand metodudur. Aşağıdaki örnekte "-b" parametresi ile birlikte kullanılmış alt parametreleri listeleyen bir örnek yazılmıştır.

```
package pTest;

import java.util.List;

import com.sanalstil.pShell.pCommands.ICommandParameter;
import com.sanalstil.pShell.pCommands.cBaseCommand;

public class cTestCommandParameter_b implements ICommandParameter
{
	public void ReciveCommand(cBaseCommand _Command, List<String> _SubParameter)
	{
		_Command.PrintConsole("Komut '-b'  parametresi ile çağrılmıştır.");
		for (int i = 0; i < _SubParameter.size();i++)
		{
			_Command.PrintConsole("Alt Parametre : " + _SubParameter.get(i));
		}
	}

	public String GetCallParameter()
	{
		return "-b";
	}

	public String GetHelpString()
	{
		return "'-b' Yadım yazısı buraya yazılmalıdır.";
	}
}
```

cExitCommand.java

Aşağıda shell'den çıkmak için, cBaseCommand sınıfından türetilmiş cExitCommand sınıfının içeriği görülmektedir. Yukarıda tanımladığımız cTestCommand sınıfından farkı parametre almamasıdır. Komut direk isteneni yapmak üzere tasarlanmıştır. İstenirse parametre eklenebilir. "Exit -b -s" gibi konsola yazılan bir komutta parametreler hiçbirşey ifade etmeden ve hatada almadan direk Shell'i sonlandıracaktır.

```
package pTest;

import java.util.List;
import com.sanalstil.pShell.cShell;
import com.sanalstil.pShell.pCommandID.cCommandID;
import com.sanalstil.pShell.pCommands.cBaseCommand;
import com.sanalstil.pShell.pCommands.cParameterSubParameter;

public class cExitCommand extends cBaseCommand
{
	public cExitCommand(cShell _OwnerShell)
	{
		super(_OwnerShell, new cCommandID(0, "Exit"));
	}

	@Override
	public void InterpretCommand(List<String> _CommandParameter,
List<cParameterSubParameter> _ParameterList)
	{
		OwnerShell.ExitShell();
	}

	@Override
	public boolean IsMultiRunnable()
	{
		return false;
	}
}
```

cStartup.java

Bu sınıf geliştirdiğimiz komut arayüz programının giriş sınıfıdır. Bu sınıfın içinden main metodu çağrılarak programımız başlatılır. Shell arayüzünü oluşturabilmek için öncelikle cShell sınıfından bir instance oluşturuyoruz. Daha sonra cBaseCommand objesinden türetilmiş konsol komutlarımızın instance'larından oluşturuyoruz. Bu instance'lar oluşturulurken hangi shell instance'ına bağlanacaklarını bildirmek amacı ile komutların oluşturulma aşamasında içlerine oluşturduğumuz shell instance'ını parmetre olarak veriyoruz. Sonra bu komutun alıcısı olarak eklenecek olan parametre objelerinden birer instance oluşturuyoruz. Daha sonra bu parametrelerin komutlara bağlanabilmesi için cBaseCommand temel sınıfının bir metodu olan "Connect" metodu çağrılır ve bu metoda parametre olarak ICommandParameter sınıfından türetilmiş ve instance'ı oluşturulmuş parametre objelerini sırayla bağlayarak tüm parametreleri komuta bağlamış oluruz. Komutları shell'e, parametreleri de komutlara bağlandıktan sonra, shell, "StartShell" metodu çağrılarak başlatılır. Artık komut shell'imiz hazır ve komut dinlemededir. Hali hazırda aktif olan "Exit" ve "Test" komutlarını artık komut konsolunda kullanabiliriz. Örnekteki komutlar ve parametreler istenildiği kadar çoğaltılabilir.

```
package pStartup;

import com.sanalstil.pShell.cShell;

import pTest.cExitCommand;
import pTest.cTestCommandNonParameterRun;
import pTest.cTestCommandParameter_b;
import pTest.cTestCommand;

public class cStartup
 {

        public static void main(String[] args)
        {
                cShell __Shell  = new cShell();
                cTestCommand __Command = new cTestCommand(__Shell);
                cTestCommandNonParameterRun __a = new cTestCommandNonParameterRun();
                Boolean __Result = __Command.Connect(__a);
                cTestCommandParameter_b __B = new cTestCommandParameter_b();
                __Result = __Command.Connect(__B);
                cExitCommand __ExitCommand = new cExitCommand(__Shell);
                __Shell.StartShell();

        }

}
```