BY NAGOOR BABU

Event Handling:

In general, in GUI Applications, when we click on a button, when select an item in check boxes, radio buttons, List boxes, choice boxes ... automatically an event will be rised, where the generated event will not be handled by the respective GUI components, where the generated events are handled by some component internally called as "Listeners"[Handlers]

In Event Handling, the main intention of Listeners is to listen the events rised by the GUI Components, handle that events and generating results back to the GUI Applications.

Similarily, in spring applications, IOC Container is able to rise events when it was started, refreshed, stopped and closed. In this context, to handle the generated events Spring Framework has provided "Event Handling".

In Spring Framework, Event Handling capability is available with Application Context only, not with BeanFactory. In Spring Framework Event Handling all events are represented in the form of predefined classes in "org.springframework.context.event" package like below.

1.ContextRefreshedEvent
2.ContextStartedEvent
3.ContextStoppedEvent
4.ContextClosedEvent
5.RequestHandledEvent

Where ContextRefreshedEvent will be rised when we start ApplicationContext or when we access "refresh()" method on ApplicationContext.

Where ContextStartedEvent will be rised when we access "start()" method on ApplicationContext.

Where ContextStoppedEvent will be rised when we access "stop()" method on ApplicationContext.

CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

Where ContextClosedEvent will be rised when we access "close()" method on ApplicationContext.

Where RequestHandledEvent will be rised in web applications when request is handled in Spring web applications.

To Handle all the above Events, Spring Framework has provided a Listener in the form of "org.springframework.context.ApplicationListener" interface and it is having the following to execute when the respective event is rised.

```
public void onApplicationEvent(XXXEvent e)
```

Note: ApplicationListener is able to listen all the events bydefault, if we want to filter the events then we have to provide the respective Event type as Generic parameter.
EX: ApplicationListener<ContextStartedEvent> is able to listen only ContextStartedEvent .

In Spring applications, if we want to implement event handling then we have to use the following steps.

1.Create implementation classes for ApplicationListener interface and provide implementation for onApplicationEvent(--) method.
2.Configure all implementation classes as bean components in spring configuration file.

Note: To perform Event Handling in spring applications we have to use "org.springframework.context.ConfigurableApplicationContext" container , it is a chaild interface to ApplicationContext interface.

Example:

ContextRefreshedListenerImpl.java

```
package com.durgasoft.listeners;
```


```
import org.springframework.context.ApplicationListener;
import org.springframework.context.event.ContextRefreshedEvent;

public class ContextRefreshedListenerImpl implements
ApplicationListener<ContextRefreshedEvent>{

   @Override
   public void onApplicationEvent(ContextRefreshedEvent e) {
      System.out.println("Application Context is Refreshed");
   }

}
```

ContextStartedListenerImpl.java

```
package com.durgasoft.listeners;

import org.springframework.context.ApplicationListener;
import org.springframework.context.event.ContextStartedEvent;

public class ContextStartedListenerImpl implements
ApplicationListener<ContextStartedEvent>{

   @Override
   public void onApplicationEvent(ContextStartedEvent e) {
      System.out.println("Application Context Started....");
   }
}
```

ContextStoppedListenerImpl.java

```
package com.durgasoft.listeners;

import org.springframework.context.ApplicationListener;
```


```
import org.springframework.context.event.ContextStoppedEvent;

public class ContextStoppedListenerImpl implements
ApplicationListener<ContextStoppedEvent>{

   @Override
   public void onApplicationEvent(ContextStoppedEvent e) {
      System.out.println("Application Context Stopped");
   }

}
```

ContextClosedListenerImpl.java

```
package com.durgasoft.listeners;

import org.springframework.context.ApplicationListener;
import org.springframework.context.event.ContextClosedEvent;

public class ContextClosedListenerImpl implements
ApplicationListener<ContextClosedEvent>{

   @Override
   public void onApplicationEvent(ContextClosedEvent e) {
      System.out.println("Application Context has Closed");
   }

}
```

HelloBean.java

```
package com.durgasoft.beans;
public class HelloBean {
   private String uname;
```


```
   public String getUname() {
      return uname;
   }

   public void setUname(String uname) {
      this.uname = uname;
   }
   public String wish(){
      return "Hello "+uname+"!";
   }
}
```

applicationContext.xml

```
<beans>
        <bean id="helloBean" class="com.durgasoft.beans.HelloBean">
                <property name="uname" value="Durga"/>
        </bean>
        <bean id="contextRefreshedEvent"
        class="com.durgasoft.listeners.ContextRefreshedListenerImpl"/>
        <bean id="contextStartedEvent"
        class="com.durgasoft.listeners.ContextStartedListenerImpl"/>
        <bean id="contextStoppedEvent"
        class="com.durgasoft.listeners.ContextStoppedListenerImpl"/>
        <bean id="contextClosedEvent"
        class="com.durgasoft.listeners.ContextClosedListenerImpl"/>
</beans>
```

Test.java

```
package com.durgasoft.test;

import com.durgasoft.beans.HelloBean;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
```


```
public class Test {
   public static void main(String[] args)throws Exception {
      ConfigurableApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
      HelloBean hello=(HelloBean)context.getBean("helloBean");
      System.out.println(hello.wish());
      context.start();
      context.refresh();
      context.stop();
      context.close();
   }
}
```

Custom Events in Spring Applications:

Custom Events are user defined events which are defined by the developers as per their application requirements.

To manage custom Events in Spring applications we have to use the following steps.

1.Create User defined Event class
2.Create User defined event Publisher class
3.Create Event Handler class
4.Configure Event Publisher class and Event Handler class in spring configuration file.
5.Create Bean components as per the appl requirements and publish events
6.Create Test application and Execute Test application.

1.Create User Event Class:

a)Declare an user defined class.
b)Extend org.springframework.context.ApplicationEvent abstract class to user defined class.
c)Declare public and Object parameterized Constructor and access super class Object parameterized constructor by using "super" keyword.

d)Define other methods as per the requirment in Event class.

2.Create User defined event Publisher class

The main intention of event publisher class is to publish the user defined event inorder to handle.

Steps:

a)Declare an user defined class.
b)Implement org.springframework.context.ApplicationEventPublisherAware interface in event class.
c)Provide implementation for setApplicationEventPublisher(--) method inorder to inject ApplicationEventPublisher object.
Note: The main intention to implement ApplicationEventPublisherAware interface is to inject ApplicationEventPublisher object only.
d)Define a method to publish an event by using the following method from ApplicationEventPublisher .
public void publishEvent(ApplicationEvent ae)

3.Create User defined Event Handler Class:

The main intention of User defined Event Handler class is to handle the user defined Events.

Steps:

a)Create an User defined class
b)Implement org.springframework.context.ApplicationListener interface.
c)Implement onApplicationEvent(--) Method in user defined class with an application logic.
Note: onApplicationEvent(--) method is able to take the parameter which is specified as generic type to ApplicationListener interface.

Note: By default, Listeners are able to handle all the Listeners, but, if we want to filter the Listeners then we have to use Generic type to ApplicationListener.

Note: In Spring Event Handling, bydefault, ApplicationContext is able to handle the events synchronously, but, if we want to handle the events Asynchronously then we have to use ApplicationEventMustcaster interface.

Note: Spring4.2 version has provided very good annotations support for Event Handling in the form of the following Annotations.
1)@EventListener({Event1.class,Event2.class...})
Where Event1.class, Event2.class,... are Event class types wich we want to process.
EX:@EventListener({ContextRefreshedEvent.class,ContextStoppedEvent.class})
2)@Async() : It will be used to prcess events asynchronously.

4)Prepare Bean components and publish events:

In the application, as per the requirement we are able to publish the events by using publishEvent(--) method.

AccountEvent.java

```
package com.durgasoft.events;

import java.io.FileOutputStream;
import java.util.Date;
import org.springframework.context.ApplicationEvent;

public class AccountEvent extends ApplicationEvent{
   static FileOutputStream fos;
   static{
      try {
         fos=new FileOutputStream("E:/logs/log.txt", true);

      } catch (Exception e) {
         e.printStackTrace();
      }
   }
   private String message;
   public AccountEvent(Object obj, String message) {
```

```
    super(obj);
    this.message=message;
  }

  public void generateLog(){
    //System.out.println("********"+message+"*******");
    try {
      message=new Date().toString()+":"+message;
      message=message+"\n";
      byte[] b=message.getBytes();
      fos.write(b);
    } catch (Exception e) {
      e.printStackTrace();
    }

  }
}
```

AccountEventPublisher.java

```
package com.durgasoft.events;

import org.springframework.context.ApplicationEventPublisher;
import org.springframework.context.ApplicationEventPublisherAware;

public class AccountEventPublisher implements ApplicationEventPublisherAware{
  private ApplicationEventPublisher publisher;
  @Override
  public void setApplicationEventPublisher(ApplicationEventPublisher publisher) {
    this.publisher=publisher;
  }
  public void publish(String message){
    AccountEvent ae=new AccountEvent(this, message);
    publisher.publishEvent(ae);
  }
}
```

AccountEventHandler.java

```
package com.durgasoft.events;

import org.springframework.context.ApplicationListener;

public class AccountEventHandler implements ApplicationListener<AccountEvent> {

   @Override
   public void onApplicationEvent(AccountEvent e) {
      e.generateLog();
   }

}
```

Account.java

```
package com.durgasoft.beans;

import com.durgasoft.events.AccountEventPublisher;

public class Account {
   private  AccountEventPublisher publisher;
   public void setPublisher(AccountEventPublisher publisher){
      this.publisher=publisher;
   }
   public void createAccount(){
      System.out.println("Account Created");
      publisher.publish("AccountCreated");
   }
   public void searchAccount(){
      System.out.println("Account Identified");
      publisher.publish("AccountIdentified");
   }
   public void updateAccount(){
```


```
        System.out.println("Account Updated");
        publisher.publish("AccountUpdated");
    }
    public void deleteAccount(){
        System.out.println("Account Deleted");
        publisher.publish("AccountDeleted");
    }
}
```

applicationContext.xml

```
<beans>
    <bean id="account" class="com.durgasoft.beans.Account">
        <property name="publisher" ref="accountEventPublisher"/>
    </bean>

    <bean id="accountEventHandler"
class="com.durgasoft.events.AccountEventHandler"/>
    <bean id="accountEventPublisher"
class="com.durgasoft.events.AccountEventPublisher"/>
</beans>
```

Test.java

```
package com.durgasoft.test;

import com.durgasoft.beans.Account;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {
    public static void main(String[] args)throws Exception {
        ConfigurableApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
        Account account=(Account)context.getBean("account");
        account.createAccount();
```

```
        account.searchAccount();
        account.updateAccount();
        account.deleteAccount();
    }

}
```

Internationalization in SPring:

Designing java applications w.r.t Local Users is called as Internationalization.

To provide Internationalization services to the users, first, we have to devide all the users into groups as per locality, for this, we have to use the following parameters.

1.language: It able to represent two lower case letters.
EX: en, it, hi, ...

2.country: It will be represented in the form of two Upper case letters.
EX: US, IN, IT,.....

3.System Varient[OS]: It will be represented in the form of three lower case letters.
EX: win, uni, lin,.....

In java applications, to represent a group of local users JAVA has provided a predefined class in the form of "java.util.Locale".

To create Locale class object we have to use the following Constructors.

public Locale(String lang)
public Locale(String lang, String country)
public Locale(String lang, String country, String sys_Varient)

EX:

```
Locale l1=new Locale("en");
```

Locale l2=new Locale("en", "US");
Locale l3=new Locale("en", "US", "win");

In Java applications, we are able to provide the following services as part of Internationalization.

1.Number Formations
2.Date Formations
3.Message Formations

1. Number Formations:

It can be used to represent a number w.r.t a particular Locale. It will use java.text.NumberFormat class to represent a number.

Steps:

a)Create Locale object.
b)Create NumberFormat class object by using getInstance() Factory method.
c)Represent Number as per the Locale by using format(-) method.

2.Date Formations:

It can be used to represent a Date w.r.t a particular Locale, for this, it will use java.text.DateFormat class.

Steps:

a)Create Locale object.
b)Create DateFormat class object by using getDateInstance(--) Factory method.
c)Represent Date w.r.t the Locale by using format(--) method.

3.Message Formations:

It can be used to represent messages w.r.t a particular Locale, for this, we have to use properties files and java.util.ResourceBundle class.

Steps:

a)Create properties files with all the messages in the form of key-value pairs.
Note: properties files names must be provided in the followng format.
baseName_lang_country.properties
b)Create ResourceBundle object by using getBundle(--) Factory method.
c)Get Message from ResourceBundle object by using getString(-) method.

Example:

com/durgasoft/resources/abc_en_US.properties

```
-------------------------------------------
welcome = Welcome To en US Users.
```

com/durgasoft/resources/abc_it_IT.properties

```
-------------------------------------------
welcome = Welcome To it IT Users.
```

Test.java

```
package com.durgasoft;
import java.text.DateFormat;
import java.text.NumberFormat;
import java.util.Date;
import java.util.Locale;
import java.util.ResourceBundle;
public class Test {
   public static void main(String[] args)throws Exception {
      Locale l = new Locale("it", "IT");

      NumberFormat num_Format = NumberFormat.getInstance(l);
      System.out.println(num_Format.format(1234567.23456));

      DateFormat date_Format = DateFormat.getDateInstance(0, l);
      System.out.println(date_Format.format(new Date()));
```


```
        ResourceBundle resource_Bundle =
ResourceBundle.getBundle("com/durgasoft/resources/abc", l);
        System.out.println(resource_Bundle.getString("welcome"));
    }
}
```

To provide Message formations in Spring applications, Spring has provided a predefined interface in the form of "org.springframework.context.MessageSource" .

For MessageSource interface Spring Framework has provided the following two implementation classes.

org.springframework.context.support.ResourceBundleMessageSource
org.springframework.context.support.ReloadableResourceBundleMessageSource

Where ResourceBundleMessageSource is able to get messages from properties files on the basis of the provided locale.

Where ReloadableResourceBundleMessageSource is able to get messages from both properties files and from XML files.

Steps:

1.Declare properties files with the messages and with the following format for properties files names.
    baseName_lang_Country.properties.

2.Declare a Bean class with MessageSource type property and the respective setter method and the required business methods.
Note: To get a message from MessageSource object we have to use the following method.
 public String getMessage(String key, Object[] place_holder_values, Locale l)

Where "key" is key of the message defined in properties file.

Where "Object[] " must be provided to provide values to the place holders which we defined in messages in properties files, if place holders are not existed in messages then we have to provide "null" value as Object[].
Where Locale is able to repersent the constants like US, FRANCE, IN,... from Locale class inorder to recognize the properties file.

3.Configure bean class in properties file and inject either ResourceBundleMessageSource or ReloadableResourceBundleMessageSource object as reference in Bean object and provide bease name as property for MessageSource object.

4.In Main class, in main(), get Bean object and access business method.

Example-1:

abc_en_US.properties

```
welcome = Welcome To {0} and {1} User.
```

abc_fr_FR.properties

```
welcome = Welcome To {0} and {1} Users.
```

I18NBean.java

```
package com.durgasoft.beans;

import java.util.Locale;
import org.springframework.context.MessageSource;

public class I18NBean {
   private MessageSource messageSource;

   public void setMessageSource(MessageSource messageSource) {
      this.messageSource = messageSource;
   }
```


```
   public void displayMessage(){
        System.out.println("Message  :"+messageSource.getMessage("welcome", new
Object[]{"fr", "FRANCE"}, Locale.FRANCE));
      System.out.println("Message  :"+messageSource.getMessage("welcome", new
Object[]{"en", "US"}, Locale.US));

   }
}
```

applicationContext.xml

```
<beans>
 <bean id="i18nBean" class="com.durgasoft.beans.I18NBean">
    <property name="messageSource" ref="resourceBundleMessageSource"/>
 </bean>
 <bean id="resourceBundleMessageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="com/durgasoft/resources/abc"/>
 </bean>
</beans>
```

Test.java

```
package com.durgasoft.test;

import com.durgasoft.beans.I18NBean;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {
   public static void main(String[] args)throws Exception {
      ApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
      I18NBean bean = (I18NBean)context.getBean("i18nBean");
      bean.displayMessage();
```


```
   }
}
```

If we want to take messages from xml file by using ReloadableResourceBundleMessageSource then we have to use define xml files with the name like baseName_lang_Country.xml and with the following tags to represent messages.

```
<properties>
 <entry key="message_Key"> Message_Value </entry>
 -----
 -----
</properties>
```

In XML files we must provide the following DTD definition.

```
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
```

Example:

abc_en_US.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
   <entry key="welcome"> Welcome to en US User from XML </entry>
</properties>
```

abc_fr_FR.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
   <entry key="welcome"> Welcome to fr France User from XML  </entry>
</properties>
```

spring_beans_config.xml

```
<beans>
   <bean id="i18nBean" class="com.durgasoft.beans.I18NBean">
      <property name="messageSource"
ref="reloadableResourceBundleMessageSource"/>
   </bean>
   <bean id="reloadableResourceBundleMessageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessageSourc
e">
      <property name="basename" value="com/durgasoft/resources_xml/abc"/>
   </bean>
</beans>
```

I18NBean.java

```
package com.durgasoft.beans;

import java.util.Locale;
import org.springframework.context.MessageSource;

public class I18NBean {
   private MessageSource messageSource;

   public void setMessageSource(MessageSource messageSource) {
      this.messageSource = messageSource;
   }

   public void displayMessage(){
      System.out.println("Message  :"+messageSource.getMessage("welcome", null,
Locale.FRANCE));
      System.out.println("Message  :"+messageSource.getMessage("welcome", null,
Locale.US));
   }
}
```

Test.java

```
package com.durgasoft.test;

import com.durgasoft.beans.I18NBean;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {
   public static void main(String[] args)throws Exception {
      ApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
      I18NBean bean = (I18NBean)context.getBean("i18nBean");
      bean.displayMessage();
   }
}
```

Bean Manipulations and Bean Wrappers:

In Bean Manipulation, we are able to perform the following actions.
1.Getting Beans Information explicitly like properties and their setXXX() and getXXX() methods.
2.Creating JavaBean Objects, checking bean property types, copying bean properties, etc.
3.Accessing fields without standard getters and setters.
4.analyze and manipulate standard JavaBeans like to get and set property values, get property descriptors, and query the readability/writability of properties and setting of index properties.

In Java, we are able to get beans descriptions like bean properties information like their names and the correspecding setXXX() and getXXX() methods information by using "Beans Introspection".

If we want to get beans data explicitly then we have to java.beans.BeanInfo interface, to get BeanInfo object then we have to use the following method from java.beans.Introspector class.

public BeanInfo getBeanInfo(Class bean_class_type)
EX: BeanInfo beanInfo = Interospector.getBeanInfo(MyBean.class);

To get All properties information of the Bean object we have to use "java.beans.PropertyDescriptor" class. To get all properties description in the form of PropertyDescriptor objects in an array then we have to use the following method from BeanInfo .

public PropertyDescriptor[] getPropertyDescriptors()
EX: PropertyDescriptor[] props = beanInfo.getPropertyDescriptors();

Example:

```
package com.durgasoft.core;
public class Employee {
   private int eno;
   private String ename;
   private float esal;
   private String eaddr;

   setXXX() and getXXX()

}
```

Test.java

```
package com.durgasoft.core;

import java.beans.BeanInfo;
import java.beans.Introspector;
import java.beans.PropertyDescriptor;
```

BY NAGOOR BABU

```
public class Test {
   public static void main(String[] args)throws Exception {
      BeanInfo beanInfo = Introspector.getBeanInfo(Employee.class);
      PropertyDescriptor[] property_desc = beanInfo.getPropertyDescriptors();
      for(PropertyDescriptor p: property_desc){
         System.out.println(p);
      }
         MethodDescriptor[] meths = beanInfo.getMethodDescriptors();
      for(MethodDescriptor m: meths){
         System.out.println(m.getName());
      }
   }

}
```

In Spring framework, to create beans and to manipulate beans explicitly Spring Framework has provided predefined library in the form of "org.springframework.beans" .

In spring "org.springframework.beans" package has provided the following classes and interfaces to perform manipulations on beans.

BeanInfoFactory: It is an alternative to "Beans Introspection" provided by Spring Framework, it can be used to get details about the Bean objects like properties details, events details,.... by using java.beans.BeanInfo object internally . Spring Framework has provided a seperate predefined implementation class for BeanInfoFactory interface in the form of "org.springframework.beans.ExtendedBeanInfoFactory" class.

To get BeanInfo object we have to use the following method from BeanInfoFactory interface.

public BeanInfo getBeanInfo(Class bean_Class_Type)

Note: BeanInfoFactory implementation, org.springframework.beans.ExtendedBeanInfoFactory, accepts JavaBeans "non-standard" setter methods as 'writable' which returns some values instead of void.

Example:

Employee.java

```
package com.durgasoft.beans;
public class Employee {
   private int eno;
   private String ename;
   private float esal;
   private String eaddr;

   public int getEno() {
      return eno;
   }

   public int setEno(int eno) {
      this.eno = eno;
      return eno;
   }

   public String getEname() {
      return ename;
   }

   public void setEname(String ename) {
      this.ename = ename;
   }

   public float getEsal() {
      return esal;
   }
```

```
  public void setEsal(float esal) {
    this.esal = esal;
  }

  public String getEaddr() {
    return eaddr;
  }

  public void setEaddr(String eaddr) {
    this.eaddr = eaddr;
  }

}
```

Test.java

```
package com.durgasoft.test;

import com.durgasoft.beans.Employee;
import java.beans.BeanInfo;
import java.beans.PropertyDescriptor;
import org.springframework.beans.BeanInfoFactory;
import org.springframework.beans.ExtendedBeanInfoFactory;

public class Test {
  public static void main(String[] args)throws Exception {
    BeanInfoFactory factory = new ExtendedBeanInfoFactory();
    BeanInfo bean_Info = factory.getBeanInfo(Employee.class);
    System.out.println(bean_Info);
    PropertyDescriptor[] props = bean_Info.getPropertyDescriptors();
    for(PropertyDescriptor p: props){
      System.out.println(p);
    }
      MethodDescriptor[] meths = beanInfo.getMethodDescriptors();
    for(MethodDescriptor m: meths){
      System.out.println(m.getName());
```

```
    }
  }

}
```

1.BeanWrapper: BeanWrapper provides methods to create Bean objects explicitly , to analyze and manipulate standard JavaBeans like the ability to get and set property values, get property descriptors and checks the readability/writability of properties. BeanWrapper is also supports setting of index properties.

Spring Framework has provided a seperate predefined implementation class for BeanWrapper in the form of "BeanWrapperImpl".

To set values to the Bean object through Bean Wrapper class we have to use the following method from BeanWrapper class.

public void setPropertyValue(String prop_Name, Object value)

Note: If we want to set all the properties at a time to Bean object, first, we have to set property names and their values in the form of Map object then set that Map object to BeanWrapper object, for this, we have to use the following method.

public void setPropertyValues(Map map)

To get property value explicitly from Bean object we have to use the following method from BeanWrapper class.

public Object getProperty(String name)

To get Bean object explicitly through BenWrapper we have to use the following method from BeNWrapper.

public Object getWrappedInstance()

BY NAGOOR BABU

To copy the properties values from one Bean object to another Bean object we have to use the following method from "org.springframework.beans.BeanUtils" class.

public void copyProperties(Object source, Object target)
Where Source object and target objects may be the objects of Same class or different classes having same property names and same property data types.

To Check whether the property is readable or writable then we have to use the following methods from BeanWrapper class.

public boolean isReadableProperty(String prop_Name)
public boolean isWritableProperty(String prop_Name)

Example:

Employee.java

```
package com.durgasoft.beans;
public class Employee {
   private int eno;
   private String ename;
   private float esal;
   private String eaddr;

   public int getEno() {
      return eno;
   }

   public void setEno(int eno) {
      this.eno = eno;
   }

   public String getEname() {
      return ename;
```


```
  }

  public void setEname(String ename) {
    this.ename = ename;
  }

  public float getEsal() {
    return esal;
  }

  public void setEsal(float esal) {
    this.esal = esal;
  }

  public String getEaddr() {
    return eaddr;
  }

  public void setEaddr(String eaddr) {
    this.eaddr = eaddr;
  }

  public void displayEmpDetails(){
    System.out.println("Employee Details");
    System.out.println("-----------------");
    System.out.println("Employee Id     :"+eno);
    System.out.println("Employee Name   :"+ename);
    System.out.println("Employee Salary :"+esal);
    System.out.println("Employee Address:"+eaddr);

  }
}
```

Test1.java

```
package com.durgasoft.test;

import com.durgasoft.beans.Employee;
import org.springframework.beans.BeanWrapper;
import org.springframework.beans.BeanWrapperImpl;

public class Test1 {
   public static void main(String[] args)throws Exception {

      BeanWrapper bw = new BeanWrapperImpl(Employee.class);
      bw.setPropertyValue("eno", 111);
      bw.setPropertyValue("ename", "AAA");
      bw.setPropertyValue("esal", 5000.0f);
      bw.setPropertyValue("eaddr", "Hyd");

      Employee emp = (Employee) bw.getWrappedInstance();
      System.out.println(emp);
      emp.displayEmpDetails();
        System.out.println();

        Map<String, String> map = new HashMap<String, String>();
      map.put("eno", "222");
      map.put("ename", "BBB");
      map.put("esal", "6000");
      map.put("eaddr", "Hyd");
      bw.setPropertyValues(map);
      System.out.println(emp);
      emp.displayEmpDetails();


        System.out.println("Employee Details");
        System.out.println("-------------------------");
        System.out.println("Employee No     :"+bw.getPropertyValue("eno"));
        System.out.println("Employee Name   :"+bw.getPropertyValue("ename"));
```

```
        System.out.println("Employee Salary  :"+bw.getPropertyValue("esal"));
        System.out.println("Employee Address :"+bw.getPropertyValue("eaddr"));

    }

}
```

Test2.java

```
package com.durgasoft.test;

import com.durgasoft.beans.Employee;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.BeanWrapper;
import org.springframework.beans.BeanWrapperImpl;

public class Test2 {
    public static void main(String[] args)throws Exception {
        Employee emp1 = new Employee();
        BeanWrapper bw = new BeanWrapperImpl(emp1);
        bw.setPropertyValue("eno", 111);
        bw.setPropertyValue("ename", "AAA");
        bw.setPropertyValue("esal", 5000.0f);
        bw.setPropertyValue("eaddr", "Hyd");
        System.out.println(emp1);
        emp1.displayEmpDetails();

        Employee emp2 = new Employee();
        BeanUtils.copyProperties(emp1, emp2);
        System.out.println(emp2);
        emp2.displayEmpDetails();

    }
}
```