CollectionMerging

 => It should be used along with BeanInheritance
 => It can be applied only on collection/array type bean properties.
 => It is all about adding more values collection/array type property in the child bean cfg by inheriting from parent bean cfg.

applicationContext.xml
======================
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd">

      <bean id="firstYear" class="in.ineuron.comp.EnggCourse" abstract="true">
            <property name="subjects">
                  <set>
                        <value>M1</value>
                        <value>CAD</value>
                        <value>Chemistry</value>
                  </set>
            </property>
      </bean>

      <bean id="CS" class="in.ineuron.comp.EnggCourse" parent="firstYear">
            <property name="subjects">
                  <set merge="true">
                        <value>DS</value>
                        <value>M3</value>
                        <value>C++</value>
                  </set>
            </property>
      </bean
</beans>
```

Output
======
```
EnggCourse [subjects=[M1, CAD, Chemistry, DS, M3, C++]]
```

Keypoints

=> Collection merging is possible only with collection, array bean properties... not on simple, object type bean properties
=> merge attribute is available only in <list>,<set>,<map>,<props>,<arry> tags
=> The possible values for merge attributes are
            a. false
            b. true
            c. default(default)
=> Both setter and constructor injection supports the "collection merging".

Beanalias

 => It refers to providing nicknames to beanid.
 => Useful when bean id is very lengthy to refer in multiple places.
 => Generally the practise to take the bean id is the classname, which is

relatively lengthy.
 => To give altername names to the bean id we should use allias.

```
      <bean id = "wishMessageGenrator"
class="in.ineuron.comp.WishMessageGenerator">
            <property name="date" ref="dt"/>
      </bean>
      <alias name="wishMessageGenerator" alias="wmg"/>
```

Autowiring

Assigning the dependant class object to target class object is called "Dependancy injection"/"AutoWiring".

=> Autowiring can be done in 2 ways
- a. Explicit Autowiring/Manual Injection
  - a. <property name='' ref=''/>
  - b. <constructor-arg name='' ref=''/>
- b. Autowiring/AutoInjection
  - <bean id ='' class='' autowire=''/>
  - autowire will take 3 values
    - a. byName
    - b. byType
    - c. constructor

Note: Autowiring is very useful becoz it helps in RAD(Rapid Application Development).

autowire = byName
==========
=> Performs Setter Injection
=> Container detects/finds dependent spring bean class object based on id that is matching with target class property
   name(Courier courier; <bean id ='courier' class='in.ineuron.bean.BlueDart'/>)
=> There is no possiblity of ambiguity problem becoz the bean id in IOC container are unique id's.

autowire = byType
==========
=> Perform Setter Injection
=> Container detects/finds dependant spring bean class object based on the property type/class type in the Target
   class(Courier courier;=> It is Courier type)
=> There is a possiblity of Ambiguity problem and we solve this problem by using "primary=true" in one of the
   dependent spring bean configuration.

autowire = constructor
==========
=> Performs constuctor injection using paramatereized constructor
=> Internally it follows byType approach only.
=> Ambiguity problem would arise and it can be solved using
- a. primary = true on any one of the bean.
- b. setting id of the bean name and parameter name of the constructor also same.

Note: if we keep id name and consturctor param name same and in any one of the bean

with primary = true, then
      constructor injection will happen by giving the priority for primary=true.

limitations of Autowiring
=========================
1. It is possible only on Object-type/reference-type bean properties.
2. There is a possiblity of getting Ambiguity problem.
3. It will also kill readability of Spring bean configuration file.

autowire-candidate = false
==========================
 =>If we don't want particular beans not to participate in autowiring then we use the above property
 =>It is one more solution to resolve the problem of ambiguity which would arise in byType.

```
       <!-- CONFIGURING THE DEPENDANT BEAN -->
      <bean id='bDart' class='in.ineuron.bean.BlueDart' />
      <bean id='dtdc' class='in.ineuron.bean.DTDC' autowire-candidate="false" primary="true"/> //invalid combo
      <bean id='courier' class='in.ineuron.bean.FirstFlight' autowire-candidate="false"/>


      <!-- CONFIGURING THE TARGET BEAN -->
      <bean id='fpkt' class='in.ineuron.bean.Flipkart' autowire="byType">
            <property name="regNo" value='12345' />
      </bean>
```

What is the difference b/w them?
            autowire = no => Disables the autowiring, programmer should explicity perform Autowiring.
            autowire-candidate =false => it makes the spring bean not to participate in Autowiring.

Note: The bean which is disabled through <bean id="" class="" autowire-candidate="false"/> can be used for Explicit Autowiring.

applicationContext.xml
======================
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd">


      <bean id="dtdc" class="in.ineuron.comp.DTDC"
            autowire-candidate="false" primary="true"/>

      <bean id="bDart" class="in.ineuron.comp.BlueDart"
            autowire-candidate="false" />

      <bean id="fFlight" class="in.ineuron.comp.FirstFlight" autowire-candidate="false"/>
```

```
      <!-- Configuring the Target bean -->
      <bean id="fpkt" class="in.ineuron.comp.Flipkart" autowire="no">
            <constructor-arg name="discount" value="10" />
            <constructor-arg name="courier" ref="fFlight"/>
      </bean>
</beans>



Support of I18N(Internationalization)
      Making our application work for all different locale is called I18N.
        Locale => Langauge+ Country
                  eg: en-US,en-BR,hi-IN,fr-FR,de-DE,.....


applicationContext.xml
======================
<bean id='messageSource'
class='org.springframework.context.support.ResourceBundleMessageSource'>
            <property name="basename" value='in/ineuron/common/App'/>
</bean>

App.properties
--------------
#Base properties file(English)
btn1.cap = insert
btn2.cap = update
btn3.cap = delete
btn4.cap = view

App_fr_FR.properties
--------------------
#Base properties file(French)
btn1.cap = insérer {0}
btn2.cap = mise à jour
btn3.cap = supprimer
btn4.cap = voir


App_de_DE.properties
--------------------
#Base properties file(German)
btn1.cap = Einfügung {0}
btn2.cap = aktualisieren
btn3.cap = löschen
btn4.cap = Sicht


App_hi_IN.properties
--------------------
#Base properties file(HINDI)
btn1.cap = \u0921\u093E\u0932\u0928\u093E {0}
btn2.cap = \u0905\u0926\u094D\u092F\u0924\u0928
btn3.cap = \u092E\u093F\u091F\u093E\u0928\u093E
btn4.cap = \u0926\u0947\u0916\u0928\u093E

ClientApp.java
==============
// started the container
```

```
ClassPathXmlApplicationContext applicationContext = new

      ClassPathXmlApplicationContext("in/ineuron/cfg/applicationContext.xml");

      // Prepare a Locale Object
      Locale locale = new Locale(args[0],args[1]);

      String cap1 = applicationContext.getMessage("btn1.cap", null, "msg1",
locale);
      String cap2 = applicationContext.getMessage("btn2.cap", null, "msg2",
locale);
      String cap3 = applicationContext.getMessage("btn3.cap", null, "msg3",
locale);
      String cap4 = applicationContext.getMessage("btn4.cap", null, "msg4",
locale);

      System.out.println(cap1 + " " + cap2 + " " + cap3 + " " + cap4);

      System.out.println();

      System.out.println(applicationContext.getMessage("btn1.cap",null,new
Locale("en","US")));
      System.out.println(applicationContext.getMessage("btn2.cap",null,new
Locale("hi","IN")));
      System.out.println(applicationContext.getMessage("btn2.cap",null,new
Locale("fr","FR")));

applicationContext.close();

Note: ctx.getMessage() internally will call ctx.getBean("id = messageSource");
```