Application Program Development

APD545

Instructor: Maryam Sepehrinour
Email: Maryam.Sepehrinour@SenecaPolytechnic.ca

Properties

- ✓ Properties are basically wrapper objects for JavaFX-based object attributes such as String or Integer.
  - ✓ Read/Writeable

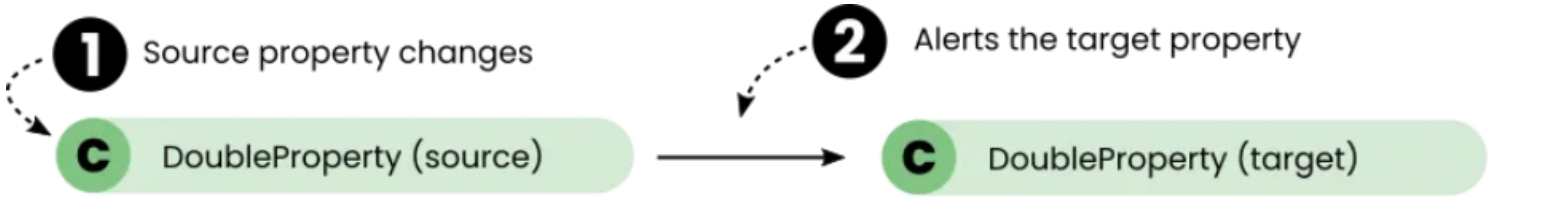
```
StringProperty password = new SimpleStringProperty("password1");
password.set("1234");
System.out.println("Modified StringProperty " + password.get() ); // 1234
```

  - ✓ Read-Only

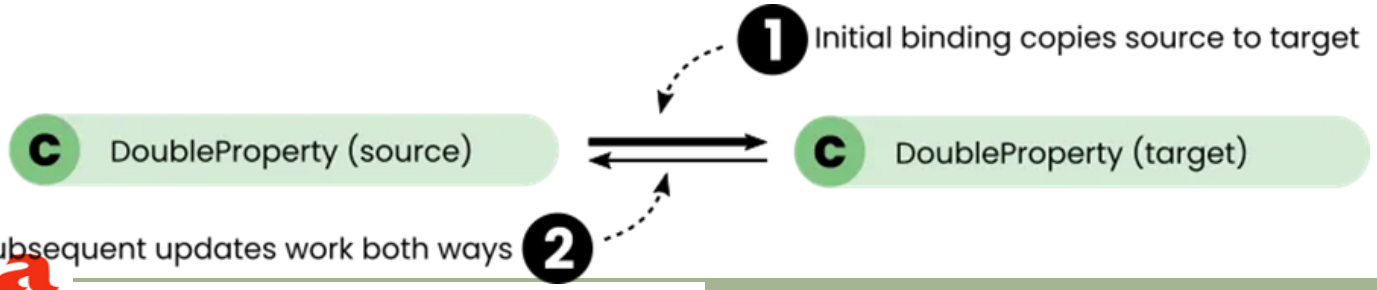
```
ReadOnlyStringWrapper userName = new ReadOnlyStringWrapper("jamestkirk");
ReadOnlyStringProperty readOnlyUserName = userName.getReadOnlyProperty();
```

Binding – What?

- ✓ Binding has the idea of at least two values (properties) being synchronized.
- ✓ This means that when a dependent variable changes, the other variable changes.
- ✓ Handling the concept by : Change-Listening
- ✓ Methods:
  - ✓bind()



  - ✓ bindBiderctional()



Binding - What?

- If the property is currently bound, the current binding is deleted and the new one replaces it
- The method immediately copies the value of the property it's listening to, so the current value of the target property is lost.
- There are 10 general binding strategies, which you can divide in two main areas

- Operations on values
- Operations on collections.
- Other (which doesn't fit any w

Values	Collections	Other bindings
Mathematics (+, - , /, x)	Binding two collections (lists, sets, maps)	Multiple-object bindings
Selecting maximum or minimum	Binding values to objects at certain position in a collection	Boolean operations (and, not, or, when)
Comparisons (=, !=, <, >, <=, >=)	Binding to collection size	Selecting values
String formatting	Whether a collection is empty	

Binding – Why?

- Purposes of Binding in JavaFx:
  - Automatic UI Updates
  - Consistency and Synchronization
  - Reduced Boilerplate Code
  - Improved Readability and Maintainability

Common Scenarios for Binding in JavaFX

- UI Control Updates
- Two-Way Data Binding: For forms and inputs
- Style and Layout Properties

Binding – How?

- ✓ Binding has the idea of at least two values (properties) being synchronized.
- ✓ This means that when a dependent variable changes, the other variable changes.
- ✓ Handling the concept by : Change-Listening
- ✓ Methods:
  - ✓bind()

```
StringProperty sourceProperty = new SimpleStringProperty("First Value");
StringProperty targetProperty = new SimpleStringProperty("Second Value");
targetProperty.bind(sourceProperty);
```

  - ✓ bindBiderctional()

```
StringProperty sourceProperty = new SimpleStringProperty("First Value");
StringProperty targetProperty = new SimpleStringProperty("Second Value");
targetProperty.bindBidirectional(sourceProperty);
```

Binding

- The Bindings API gives you four different ways to bind to a collection:
  - carbon-copy
  - index-binding
  - size-binding
  - emptiness-binding.

- Only the first one copies the contents of your collection into a target collection.

```
bindContent(List<E> list1, ObservableList<? extends E> list2)
bindContent(Map<K,V> map1, ObservableMap<? extends K,? extends V> map2)
bindContent(Set<E> set1, ObservableSet<? extends E> set2)
```

Generics - Templates

- Generics is the capability to parameterize types.
-

```
class Store{
        private Bookmark a;

        public void set(Bookmark a){
                this.a = a;
        }

        public Bookmark get(){
                return a;
        }
}
```

Can only hold **Bookmark** objects or its subtype objects

✓ **Type is hardcoded**

Generics - Templates

- Generics is the capability to parameterize types.
-

```
class Store{
        private Bookmark a;

        public void set(Bookmark a){
                this.a = a;
        }

        public Bookmark get(){
                return a;
        }
}
```

Can only hold **Bookmark** objects or its subtype objects

✓ **Type is hardcoded**

```
class Store{
        private Object a;

        public void set(Object a){
                this.a = a;
        }

        public Object get(){
                return a;
        }
}
```

Generics - Templates

- Generics is the capability to parameterize types.

```
class Store{
        priva

        publi

        }

        publi

        }
}
```

```
class Store <T>{
        private T a;

        public void set(T a){
                this.a = a;
        }

        public T get(){
                return a;
        }
}
```

```
Store{
 private Object a;

 public void set(Object a){
        this.a = a;
 }

 public Object get(){
        return a;
 }
}
```

Generics - Templates

```
class Store <T>{
	private T a;

	public void set(T a){
		this.a = a;
	}

	public T get(){
		return a;
	}
}
```

```
Store<String> stringStore = new Store<String>()
Store<Date> dateStore = new Store<Date>()
Store<List<Date> > dateListStore = new Store<List<Date>>()
Store<Book> bookStore = new Store<>()
```

Generics - Templates – Why?

Stronger Type checking :

- Fixing Error at run-time or Fixing error at compile time?

```
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(32); //Compile Time Error
```

Casting can be eliminated:

- No more object types casting.

with type cast

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

without type case

```
List <String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);   //no cast
```

Type Safety:

- Holds only single type of objects, doesn't allow to store other objects.

GUI Editor

- ✓ Please take a look at Week #6 - Segment #4, and try the mentioned items to handle some GUI concepts!

Thank you!