PGR112 – Step 10: Set, Optional, equals and hashCode

Object Oriented Programming

Bogdan Marculescu/ Bogdan.Marculescu@kristiania.no

Agenda

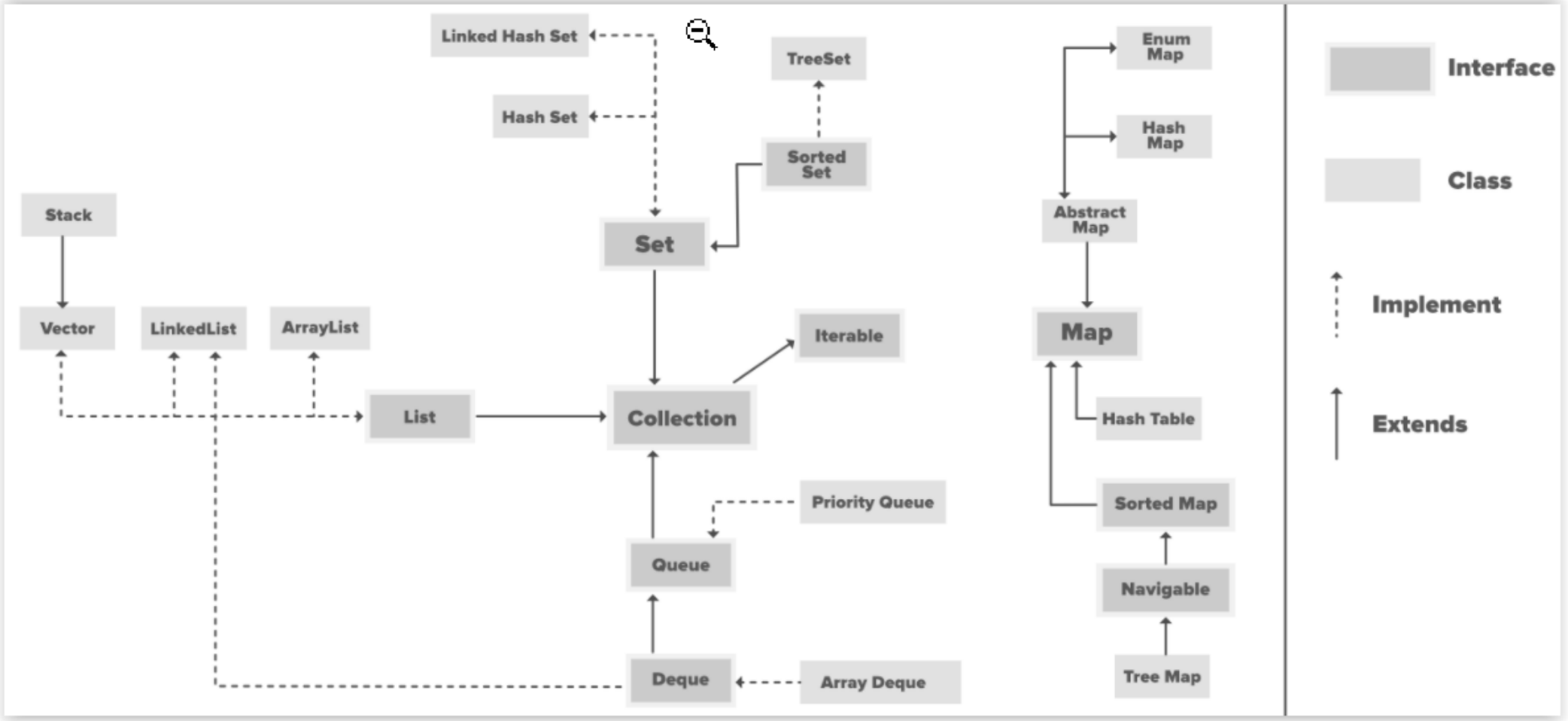
- Collection (Set)
- Optional
- HashCode
- Equals
- Validation

Status

- Step 1: Development environment. Hello World.
- Step 2: Methods (loops, if, array, String etc)
- Step 3: Classes and objects. Encapsulation.
- Step 4: ArrayList, Scanner, user input, debugger
- Step5: Read from file. Write to file. Exception handling
- Step 6: Larger task (with options)
- Step 7: Inheritance, static, final
- Step 8: Abstract classes and HashMap
- Step 9: Interface, polymorphism, casting
- **Step 10: Set, equals, hashCode, Optional, validation**
- Step 11: Larger task (with options for customization)
- Step 12: Repetition and a look at last year's exam
- Steps beyond: JDBC – Java DataBase Connector and beyond!

Collection

geeksforgeeks



Iterable

Interface Iterable<T>

Type Parameters:

T - the type of elements returned by the iterator

All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Collection<E>, Deque<E>, DirectoryStream<T>, EventSet, List<E>, NavigableSet<E>, NodeSetData<T>, Path, Queue<E>, SecureDirectoryStream<T>, Set<E>, SortedSet<E>, TransferQueue<E>, XPathNodes

All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BatchUpdateException, BeanContextServicesSupport, BeanContextSupport, ConcurrentHashMap.KeySetView, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DataTruncation, DelayQueue, DocTreePath, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, RowSetWarning, SerialException, ServiceLoader, SQLClientInfoException, SQLDataException, SQLException, SQLFeatureNotSupportedException, SQLIntegrityConstraintViolationException, SQLInvalidAuthorizationSpecException, SQLNonTransientConnectionException, SQLNonTransientException, SQLRecoverableException, SQLSyntaxErrorException, SQLTimeoutException, SQLTransactionRollbackException, SQLTransientConnectionException, SQLTransientException, SQLWarning, Stack, SyncFactoryException, SynchronousQueue, SyncProviderException, TreePath, TreeSet, Vector

Iterable, methods

Method Summary

All Methods	Instance Methods	Abstract Methods	Default Methods

Modifier and Type	Method	Description
default void	forEach (Consumer<? super T> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
Iterator<T>	iterator()	Returns an iterator over elements of type T.
default Spliterator<T>	spliterator()	Creates a Spliterator over the elements described by this Iterable.

Collection

Interface Collection<E>

Type Parameters:

E - the type of elements in this collection

All Superinterfaces:

Iterable<E>

All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, EventSet, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>, TransferQueue<E>

All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentHashMap.KeySetView, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

---

```
public interface Collection<E>
extends Iterable<E>
```

Collection, methods

All Methods	Instance Methods	Abstract M
Modifier and Type	**Method**	
boolean	add(E e)	
boolean	addAll(Collection<? extends E> c)	
void	clear()	
boolean	contains(Object o)	
boolean	containsAll(Collection<?> c)	
boolean	equals(Object o)	
int	hashCode()	
boolean	isEmpty()	
Iterator<E>	iterator()	
default Stream<E>	parallelStream()	

boolean	remove(Object o)
boolean	removeAll(Collection<?> c)
default boolean	removeIf(Predicate<? super E> filter)
boolean	retainAll(Collection<?> c)
int	size()
default Spliterator<E>	spliterator()
default Stream<E>	stream()
Object[]	toArray()
default <T> T[]	toArray (IntFunction<T[]> generator)
<T> T[]	toArray(T[] a)

Set

Interface Set<E>

Type Parameters:

E - the type of elements maintained by this set

All Superinterfaces:

Collection<E>, Iterable<E>

All Known Subinterfaces:

EventSet, NavigableSet<E>, SortedSet<E>

All Known Implementing Classes:

AbstractSet, ConcurrentHashMap.KeySetView, ConcurrentSkipListSet, CopyOnWriteArraySet, EnumSet, HashSet
JobStateReasons, LinkedHashSet, TreeSet

```
public interface Set<E>
extends Collection<E>
```

Set, methods

Method Summary

All Methods	Static Methods	Instance Methods

Modifier and Type	Method
boolean	add(E e)
boolean	addAll(Collection<? extends E> c)
void	clear()
boolean	contains(Object o)
boolean	containsAll(Collection<?> c)
static <E> Set<E>	copyOf(Collection<? extends E> coll)
boolean	equals(Object o)
int	hashCode()
boolean	isEmpty()
Iterator<E>	iterator()
static <E> Set<E>	of()
static <E> Set<E>	of(E e1)
static <E> Set<E>	of(E... elements)
static <E> Set<E>	of(E e1, E e2)
static <E> Set<E>	of(E e1, E e2, E e3)
static <E> Set<E>	of(E e1, E e2, E e3, E e4)
static <E> Set<E>	of(E e1, E e2, E e3, E e4, E e5, E e6)
static <E> Set<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7)
static <E> Set<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8)
static <E> Set<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9)
static <E> Set<E>	of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9, E e10)
boolean	remove(Object o)
boolean	removeAll(Collection<?> c)
boolean	retainAll(Collection<?> c)
int	size()
default Spliterator<E>	spliterator()
Object[]	toArray()
<T> T[]	toArray(T[] a)

HashSet

Class HashSet<E>

java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractSet<E>
            java.util.HashSet<E>

Type Parameters:

E - the type of elements maintained by this set

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, Set<E>

Direct Known Subclasses:

JobStateReasons, LinkedHashSet

```
public class HashSet<E>
extends AbstractSet<E>
implements Set<E>, Cloneable, Serializable
```

HashSet, methods

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
boolean	add(E e)	Adds the specified element to this set if it is not already present.
void	clear()	Removes all of the elements from this set.
Object	clone()	Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
boolean	contains (Object o)	Returns true if this set contains the specified element.
boolean	isEmpty()	Returns true if this set contains no elements.
Iterator<E>	iterator()	Returns an iterator over the elements in this set.
boolean	remove(Object o)	Removes the specified element from this set if it is present.
int	size()	Returns the number of elements in this set (its cardinality).
Spliterator<E>	spliterator()	Creates a *late-binding* and *fail-fast* Spliterator over the elements in this set.

HashSet, inherited methods

Methods declared in class java.util.AbstractSet

`equals, hashCode, removeAll`

Methods declared in class java.util.AbstractCollection

`addAll, containsAll, retainAll, toArray, toArray, toString`

Methods declared in class java.lang.Object

`finalize, getClass, notify, notifyAll, wait, wait, wait`

Methods declared in interface java.util.Collection

`parallelStream, removeIf, stream, toArray`

Methods declared in interface java.lang.Iterable

`forEach`

Methods declared in interface java.util.Set

`addAll, containsAll, equals, hashCode, removeAll, retainAll, toArray, toArray`

Demo, HashSet

- That’s all well and good. Shall we try them in practice?

Object

public class **Object**

Class Object is the root c
methods of this class.

All Methods	Instance Methods	Concrete Methods	Deprecated Methods

Modifier and Type	Method	Description
protected Object	clone()	Creates and returns a copy of this object.
boolean	equals(Object obj)	Indicates whether some other object is "equal to" this one.
protected void	finalize()	**Deprecated.** The finalization mechanism is inherently problematic.
Class<?>	getClass()	Returns the runtime class of this Object.
int	hashCode()	Returns a hash code value for the object.
void	notify()	Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll()	Wakes up all threads that are waiting on this object's monitor.
String	toString()	Returns a string representation of the object.
void	wait()	Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*.
void	wait (long timeoutMillis)	Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*, or until a certain amount of real time has elapsed.
void	wait (long timeoutMillis, int nanos)	Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*, or until a certain amount of real time has elapsed.

Equals

- Checks if two objects are equal. If we do not override this (from Object) it will check if two object references are the same (i.e. refer to the same object).
- We can choose to override the method. We compare to see if the objects are of the same type, and if these have the same state (i.e. The same values for their fields).

Equals, cont.

- When we override equals, then there are certain rules to keep in mind:
  - **Reflexive :** for any reference value a, a.equals(a) should return true.
  - **Symmetric :** for any reference values a and b, if a.equals(b) should return true then b.equals(a) must return true.
  - **Transitive :** for any reference values a, b, and c, if a.equals(b) returns true and b.equals(c) returns true, then a.equals(c) should return true.
  - **Consistent :** for any reference values a and b, multiple invocations of a.equals(b) consistently return true or consistently return false, provided no information used in equals comparisons on the object is modified.

HashCode

- Returns the object's hashcode value as an int.
- The value is used in collections that have to do with hashing (HashMap, HashSet, HashTable etc.).
- If we override equals, then we must override hashCode.
- If two objects are equal (based on the equals method), then they must give the same hashCode value.
- Two objects that are different (based on the equals method) must not give different hashCode value.

Demo, equals and hashCode

- IntelliJ to the rescue!

Let’s go back to HashSet - add

add

`public boolean add(E e)`

Adds the specified element to this set if it is not already present. More formally, adds the specified element `e` to this set if this set contains no element `e2` such that `Objects.equals(e, e2)`. If this set already contains the element, the call leaves the set unchanged and returns `false`.

Specified by:

`add` in interface `Collection<E>`

Specified by:

`add` in interface `Set<E>`

Overrides:

`add` in class `AbstractCollection<E>`

Parameters:

`e` - element to be added to this set

Returns:

`true` if this set did not already contain the specified element

Back to HashSet - remove

remove

```
public boolean remove(Object o)
```

Removes the specified element from this set if it is present. More formally, removes an element e such that `Objects.equals(o, e)`, if this set contains such an element. Returns `true` if this set contained the element (or equivalently, if this set changed as a result of the call). (This set will not contain the element once the call returns.)

Specified by:

`remove` in interface `Collection<E>`

Specified by:

`remove` in interface `Set<E>`

Overrides:

`remove` in class `AbstractCollection<E>`

Parameters:

`o` - object to be removed from this set, if present

Returns:

`true` if the set contained the specified element

Optional

- We have previously seen *null*, and we recognize it from the database subject.
- An object reference can refer to an object, or to null.
- Java was previously criticized for how null created an unnecessary amount of clutter / noise.
- We have to check if an object is null before we can call on methods in the object.
- If we call a method in an object that turns out to be null, we get NullPointerException…
- From Java 8 we can use Optional to improve on this.

```
public final class Optional<T>
extends Object
```

A container object which may or may not contain a non-`null` value. If a value is present, `isPresent()` returns `true`. If no value is present, the object is considered *empty* and `isPresent()` returns `false`.

Additional methods that depend on the presence or absence of a contained value are provided, such as `orElse()` (returns a default value if no value is present) and `ifPresent()` (performs an action if a value is present).

Demo, Optional

- We will not go into Optional in detail, but look at some simple mechanisms. Those who want to investigate more can, for example, read here.

Validating method inputs

- Sometimes, we will ensure that input to the method is acceptable so that the method can perform what it is supposed to. It is therefore not always sufficient to know what types of arguments there are.
- What should we do if the input is not satisfactory?
  - Throw an exception? (e.g. IllegalArgumentException)
  - Return a default value? (if the method is not void)
  - Quit the method? (if the method is void)
- Well that all depends, doesn't it?☺

Demo, validating method inputs

Some recommendations

- If you choose to deliberately throw a RuntimeException from the method (for example, IllegalArgumentException), then the exception should not be in the method's signature (but described in JavaDoc).
- You can throw exceptions from constructors as well.
- It is more important to validate input from public methods than private ones. Private methods are only called from the class itself.

Conclusion

- Goals for this step:
  - I master the use of Set, and I know when it may be appropriate.
  - I understand how I can use Optional.
  - I know when it is wise to implement equals and hashCode for objects.
  - I understand the need to validate input in methods.

Good luck with the tasks!

Next, some task, some repetition lectures, and then on to JDBC!