Answers to End of Section and Review Exercises for Chapter 6

Exercises 6.1

1. If you implement the `ArrayBag` and `ArraySortedBag` as complete, independent classes, there are several methods and one data variable that use the same code. By putting these methods in just one class, `ArrayBag`, and making this the parent class of the other class, `ArraySortedBag`, you can eliminate the duplicated code.
2. The `ArraySortedBag` must still include an `__init__` method, which calls the `__init__` method in the parent class, `ArrayBag`, with the source collection as an argument. The latter `__init__` method is not automatically called when an instance of `ArraySortedBag` is created, and its creator might be passing a source collection as an argument, which needs to be sent on to the parent class for processing.
3. When the `remove` method is called on an object of type `ArraySortedBag`, Python locates the implementation of this method in the parent class, `ArrayBag`. Because `ArraySortedBag` does not include its own implementation of this method, it becomes available via inheritance from the parent class.
4. The `ArraySortedBag` method `add` calls the `ArrayBag` method `add` when the item to insert is greater than or equal to the last item in the bag, or when the bag is empty. That's because this version of the method simply adds the new item to the end of the array in the bag, and it's just easier to call a method to do this than to repeat the code for it.
5. When the `add` method is called in the `__init__` or `__add__` method of the `AbstractBag` class, Python looks for the method's implementation in the class of the object referred to by `self`. This object is either the new type of bag just being created (in the case of `__init__`) or the left operand of the `+` operation (in the case of `__add__`). Thus, the object's class would be either `ArrayBag`, `LinkedBag`, or `ArraySortedBag`; the call here is actually "down" to a method in a class that is below `AbstractBag` in the hierarchy. Put another way, the class of the object referred to by `self` is always the class named when that object is instantiated, and any methods prefixed by `self` refer to those defined in that class if they exist there.

Exercises 6.2

1. An abstract class serves as a repository for code that looks the same in several other classes. For example, the `AbstractBag` class includes implementations of the `__str__` and `__eq__` methods for all bag classes, because these implementations never change. An abstract class is never instantiated, because it does not have the full functionality of a concrete class. This functionality usually includes the data structures that vary from concrete class to concrete class, as well as the mutator

methods that access these data. For example, the `ArrayBag` class contains an array, whereas the `LinkedBag` class contains linked nodes.

2. The `__str__` and `__eq__` methods might be redefined in the subclasses of `AbstractionCollection`. The `__str__` method might return a string with different styles of delimiters, such as square brackets or curly braces. The `__eq__` method might employ rules for determining equality that vary with the type of collections being compared.
3. These methods are `__iter__` and `add`. These methods are called by other methods in `AbtractCollection`, but they both must access the data structures in the subclasses. For example, the `add` method must know about arrays or linked nodes, but `AbstractCollection` can't have any such knowledge.
4. Here is the code:

```
def clone(self):
    return type(self)(self)
```

Answers to Review Questions

1. b
2. b
3. b
4. a
5. a
6. b
7. a