Generics

CPT204 Advanced Object-Oriented Programming

Lecture 4 Generics

What are Generics?

- ***Generics*** is the capability to ***parameterize types***
  - With this capability, you can define a class or a method with generic types that can be substituted using concrete types by the compiler
  - You may define a generic stack class that stores the elements of a generic type
    - From this generic class, you may create:
      - a stack object for holding Strings
      - a stack object for holding numbers

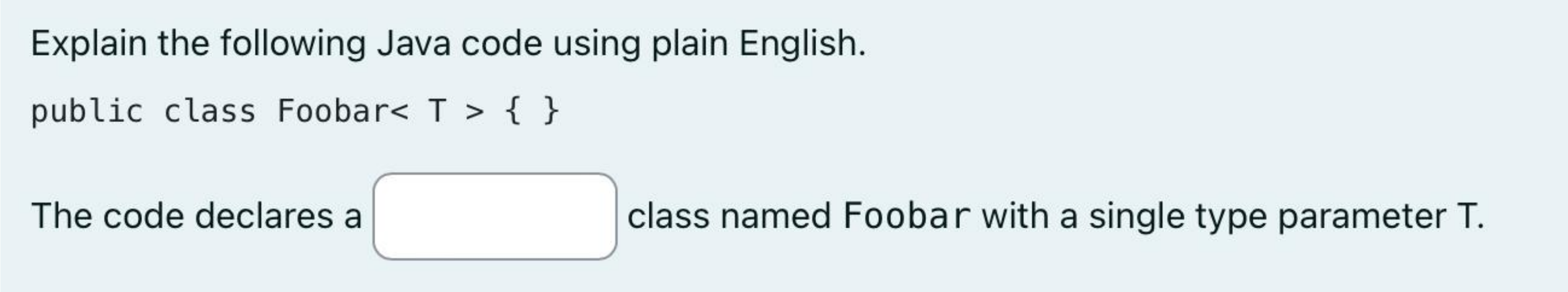
      Strings and numbers are concrete types that replace the generic type

Why Generics?

- The key benefit of generics is to enable errors to be detected at compile time rather than at runtime
  - A generic class or method permits you to specify allowable types of objects that the class or method may work with
    - We still do **code reuse**, e.g., write a single implementation for a special kind of data structure, like a single implementation of a generic stack and its standard methods
  - Most important advantage: If you attempt to use the class or method with an incompatible object, a **compile error** occurs

W4 - Sample Questions on Generic

- Conceptual & Programming
- Example 1



Explain the following Java code using plain English.

```
public class Foobar< T > { }
```

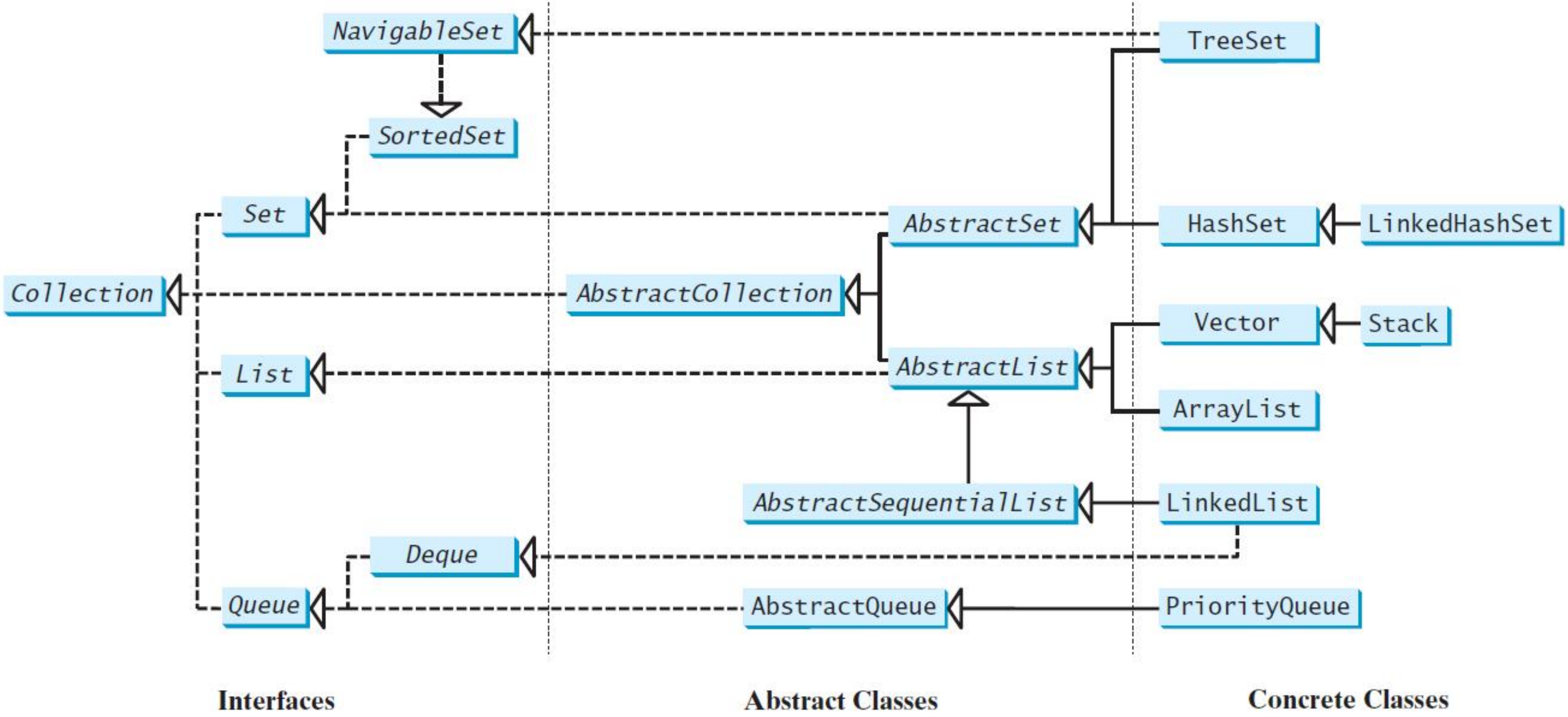
The code declares a □ class named `Foobar` with a single type parameter T.

- Example 2
  - Write Java code that declares a generic public class named Foobar with a single type parameter T.

Lists, Stacks, Queues, and Priority Queues

CPT204 Advanced Object-Oriented Programming

Lecture 5 Lists, Stacks, Queues, and Priority Queues


NavigableSet
TreeSet
SortedSet
Set
AbstractSet
HashSet
LinkedHashSet
Collection
AbstractCollection
Vector
Stack
List
AbstractList
ArrayList
AbstractSequentialList
LinkedList
Deque
Queue
AbstractQueue
PriorityQueue
Interfaces
Abstract Classes
Concrete Classes

Iterators

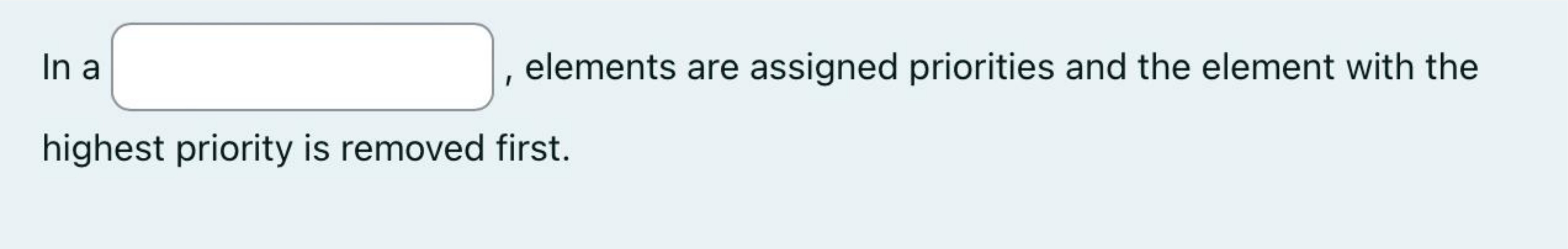
- Each collection is **`Iterable`**
  - ***Iterator*** is a classic design pattern for walking through a data structure without having to expose the details of how data is stored in the data structure
    - Also used in for-each loops:

```
for(String element: collection)
      System.out.print(element + " ");
```

- The **`Collection`** interface extends the **`Iterable`** interface
  - You can obtain a collection **`Iterator`** object to traverse all the elements in the collection with the **`iterator()`** method in the **`Iterable`** interface which returns an instance of **`Iterator`**
    - The **`Iterable`** interface defines the **`iterator`** method, which returns an **`Iterator`**

W5 - Sample Questions on Lists, Stacks, Queues, and Priority Queues

- Conceptual & Programming
- Example 1



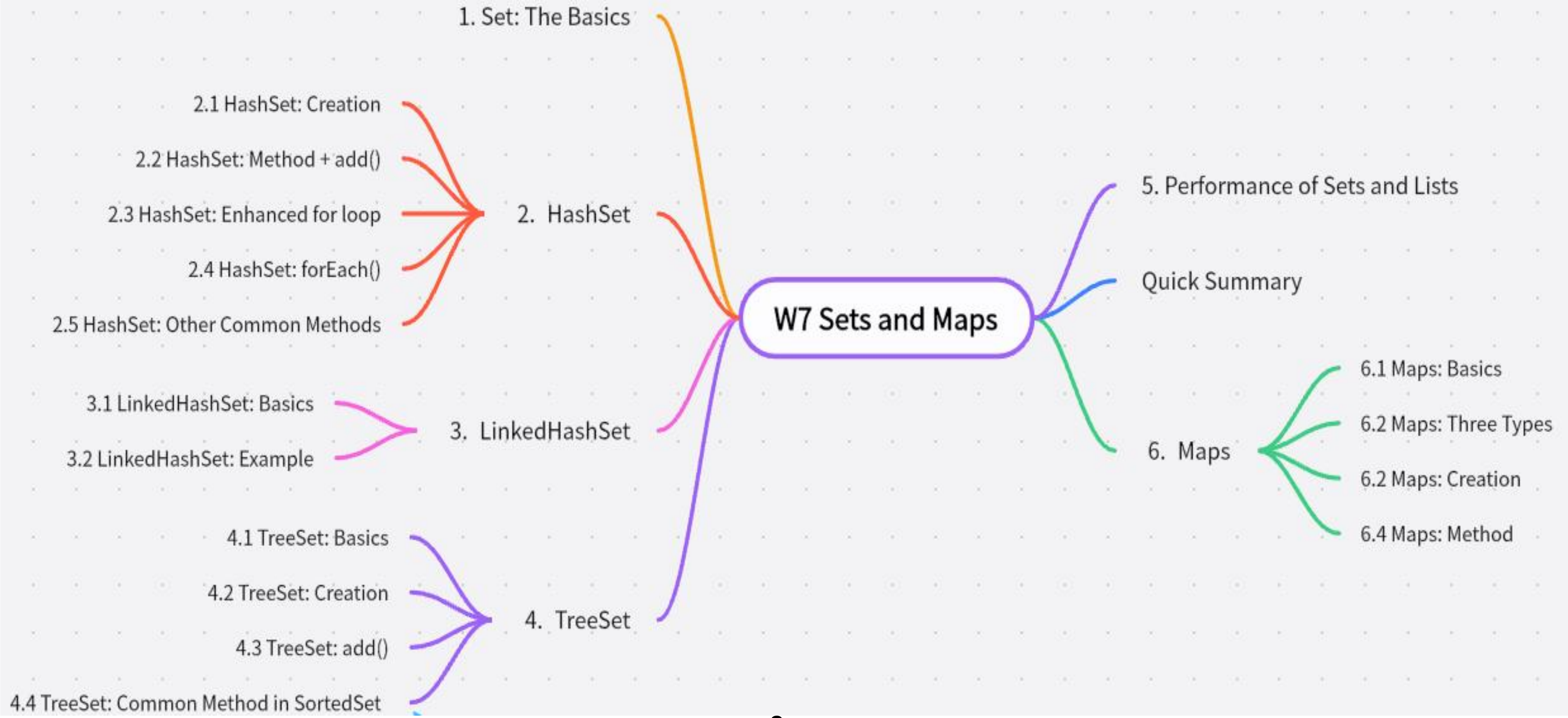
In a □ , elements are assigned priorities and the element with the highest priority is removed first.

- Example 2
  - Write Java code that (1) declares a priority queue of String type; (2) adds “A”, “a”, “1” into the priority queue; (3) output the 3 Strings.
- Just because we use priority queue as examples above, doesn't mean other containers are not important.

Sets and Maps

CPT 204 - Advanced OO Programming

Content



W7 Sets and Maps

1. Set: The Basics

2. HashSet
- 2.1 HashSet: Creation
- 2.2 HashSet: Method + add()
- 2.3 HashSet: Enhanced for loop
- 2.4 HashSet: forEach()
- 2.5 HashSet: Other Common Methods

3. LinkedHashSet
- 3.1 LinkedHashSet: Basics
- 3.2 LinkedHashSet: Example

4. TreeSet
- 4.1 TreeSet: Basics
- 4.2 TreeSet: Creation
- 4.3 TreeSet: add()
- 4.4 TreeSet: Common Method in SortedSet

5. Performance of Sets and Lists

Quick Summary

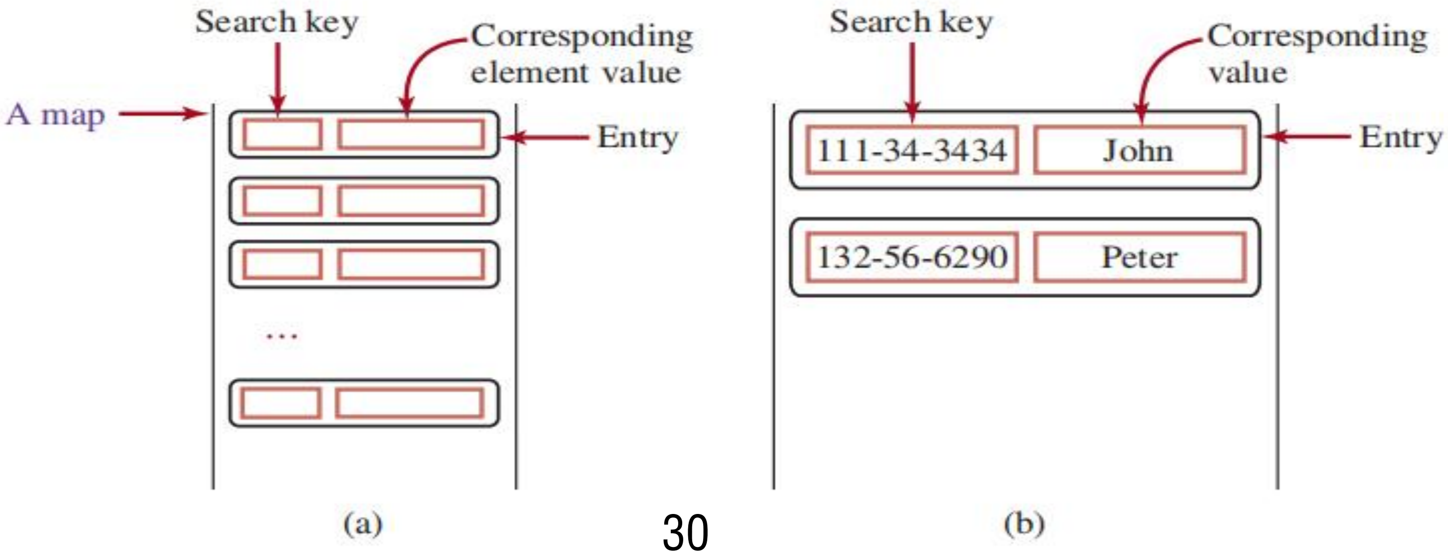
6. Maps
- 6.1 Maps: Basics
- 6.2 Maps: Three Types
- 6.2 Maps: Creation
- 6.4 Maps: Method

Set: The Basics

- **Set interface**is a sub-interface of **Collection**
- It extends the **Collection,** but does not introduce new methods or constants.
- However, the **Set interface stipulates** that an instance of **Set contains no duplicate elements**
    - That is, **no two elements e1**and **e2**can be in the set such that **e1.equals(e2)**is true

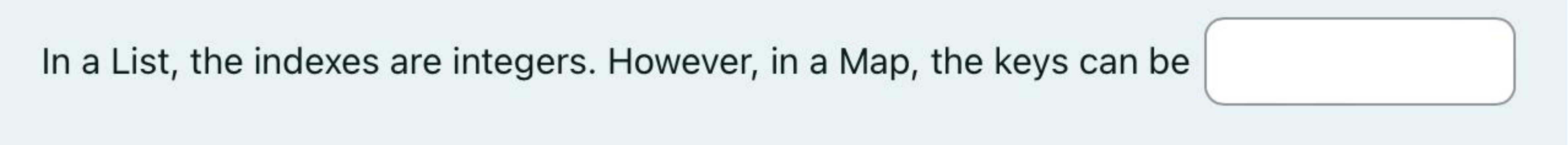
Maps: Basics

- A **map** is a container object that stores a collection of key/value pairs.
- It enables fast retrieval, deletion, and updating of the pair through the key. A map stores the values along with the keys.
- In List, the indexes are integers. In Map, the keys can be any objects.
  - A map cannot contain duplicate keys.
  - Each key maps to one value.



W6 - Sets and Maps

- Conceptual & Programming
- Example 1



In a List, the indexes are integers. However, in a Map, the keys can be

- Example 2
  - Write a Java statement that (1) declares a Hash Set of String type; (2) adds “A”, “a”, “1” into the hash set; (3) output the 3 Strings.
- Just because we used hash set as examples above, doesn't mean other containers are not important