Java Programming - 2

Sridhar A

Enum Types

- It’s a short cut for static class containing all the members with public, static and final identifiers.

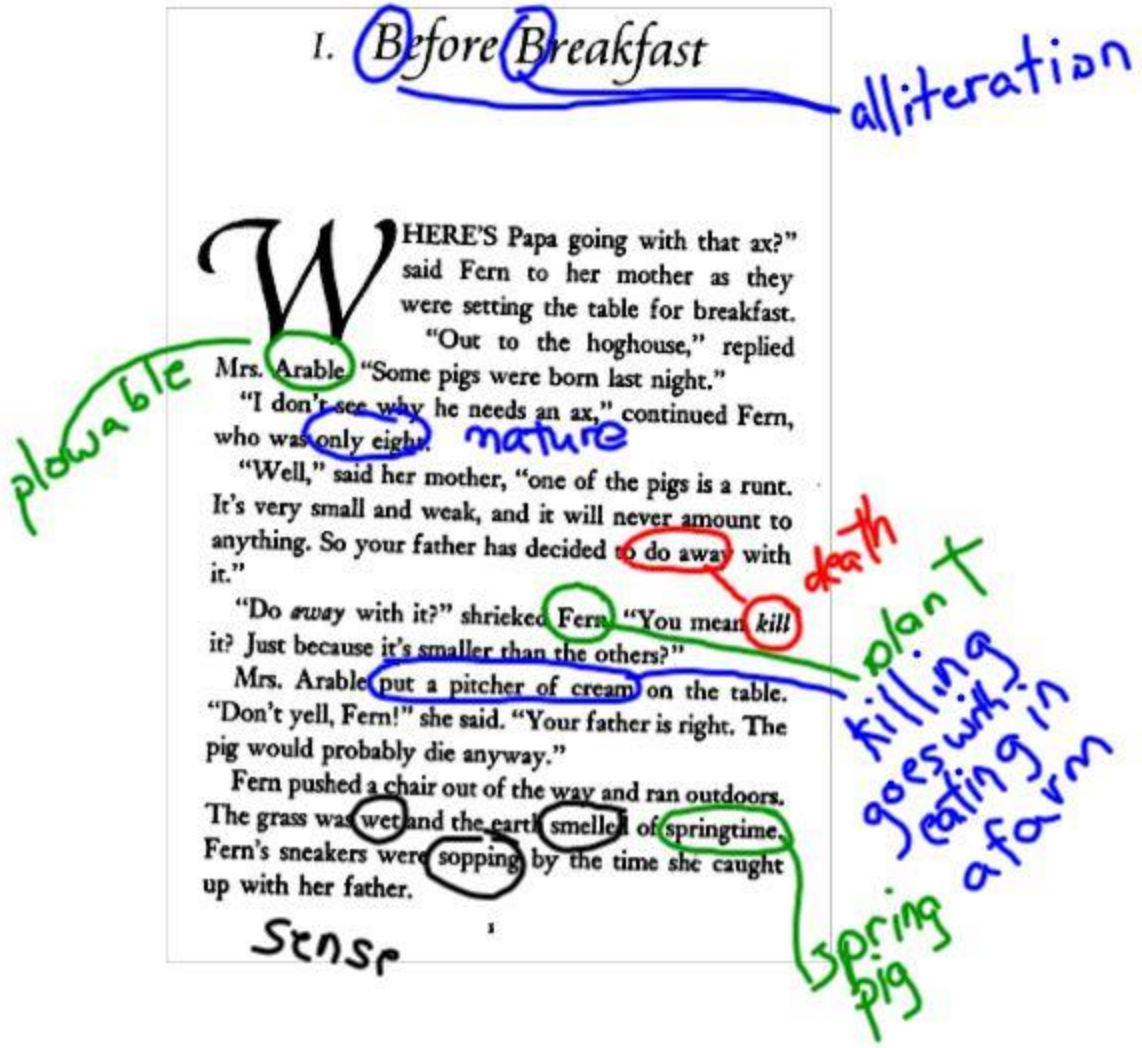
```
e.g.:
public enum Season{SPRING, SUMMER, FALL, WINTER}

public class Season2
{
    public static final int SPRING = 0;
    public static final int SUMMER = 1;
    public static final int FALL = 2;
    public static final int WINTER = 3;
}
```

Java constants are spelled in all upper-case letters

Enum Types

- Use Enum types whenever you need to define fixed set of constants.

Annotations



Annotations

- **Information for the compiler**
- **Compile-time and deployment-time processing**
- **Runtime processing**

Annotations

```
package service;

@Remotable
public interface HelloService
{
    String hello ( String message );
}
```

```
package service;

@Service ( HelloService.class )
public class HelloServiceImpl implements HelloService
{
    String hello ( String message ) {
        ...
    }
```

Annotations

```
@Entity
```

```
@Override
void mySuperMethod() { ... }
```

```
@Author(
    name = "Benjamin Franklin",
    date = "3/27/2003"
)
class MyClass() { ... }
```

```
@SuppressWarnings(value = "unchecked")
void myMethod() { ... }
```

```
@SuppressWarnings("unchecked")
void myMethod() { ... }
```

```
@Author(name = "Jane Doe")
@EBook
class MyClass { ... }
```

Annotations

As of the Java SE 8 release, annotations can also be applied to the *use* of types. Here are some examples:

- Class instance creation expression:

```
new @Interned MyObject();
```

- Type cast:

```
myString = (@NonNull String) str;
```

- `implements` clause:

```
class UnmodifiableList<T> implements
    @Readonly List<@Readonly T> { ... }
```

- Thrown exception declaration:

```
void monitorTemperature() throws
    @Critical TemperatureException { ... }
```

Declaring Annotations

```
public class Generation3List extends Generation2List {

   // Author: John Doe
   // Date: 3/17/2002
   // Current revision: 6
   // Last modified: 4/12/2004
   // By: Jane Doe
   // Reviewers: Alice, Bill, Cindy

   // class code goes here

}
```

```
@interface ClassPreamble {
   String author();
   String date();
   int currentRevision() default 1;
   String lastModified() default "N/A";
   String lastModifiedBy() default "N/A";
   // Note use of array
   String[] reviewers();
}
```

```
@ClassPreamble (
   author = "John Doe",
   date = "3/17/2002",
   currentRevision = 6,
   lastModified = "4/12/2004",
   lastModifiedBy = "Jane Doe",
   // Note array notation
   reviewers = {"Alice", "Bob", "Cindy"}
)
public class Generation3List extends Generation2List {

// class code goes here

}
```

Predefined Annotations

- **@Deprecated**
- **@Override**
- **@SuppressWarnings**
- **@Retention**
- **@Documented**
- **@Target**
- **@Remotable**
- **@Service**

Repeating Annotations

```
@Schedule(dayOfMonth="last")
@Schedule(dayOfWeek="Fri", hour="23")
public void doPeriodicCleanup() { ... }
```

```
@Alert(role="Manager")
@Alert(role="Administrator")
public class UnauthorizedAccessException extends SecurityException { ... }
```

Declaring Repeatable Annotations

```
import java.lang.annotation.Repeatable;

@Repeatable(Schedules.class)
public @interface Schedule {
  String dayOfMonth() default "first";
  String dayOfWeek() default "Mon";
  int hour() default 12;
}


public @interface Schedules {
    Schedule[] value();
}
```

Retrieving Annotations

- AnnotatedElement.getAnnotationByType(Class<T>)

More Examples

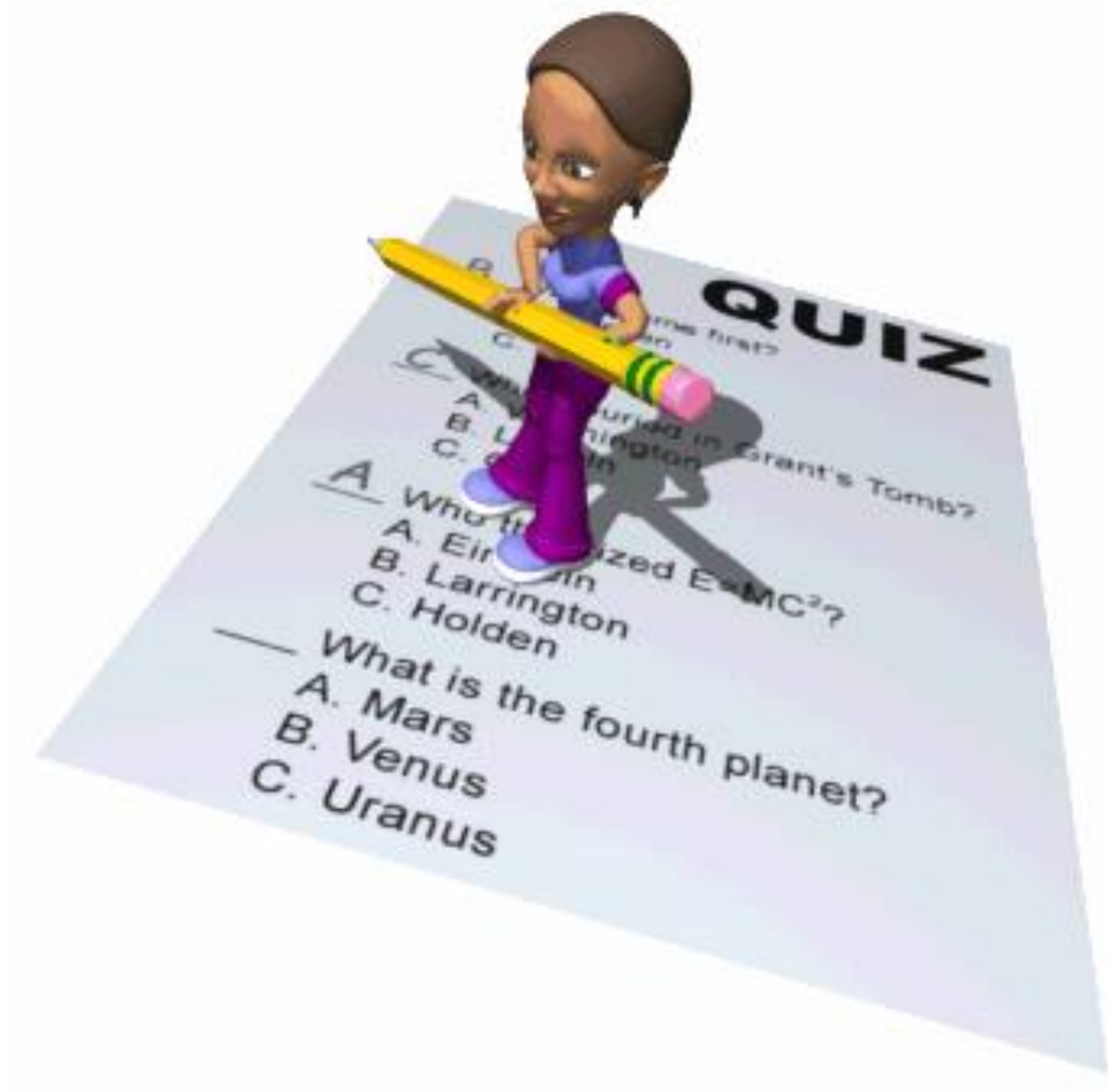
Department.java

```
package nl.amis.model.hr.entities;

import java.io.Serializable;
import javax.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

import java.util.List;
import static javax.persistence.FetchType.EAGER;

/**
 * The persistent class for the DEPARTMENTS database table.
 *
 */
@Entity
@Table(name="DEPARTMENTS")
@NamedQuery(name = "department.FindAll", query = "SELECT o FROM Department o")
public class Department implements Serializable {
    private static final long serialVersionUID = 1L;


    @NotNull
    @Size(min=2,max=30, message = "Name must between 2 and 30 characters")
    @Pattern( regexp="[A-Za-z ]*" ,message = "{departmentNameValidation}")
    @Column(name="DEPARTMENT_NAME")
    private String departmentName;
```

More Examples

*DataBeanCDI.java | DataBean.java

```
package nl.amis.web.beans;

import java.io.Serializable;

@ManagedBean( name = "dataBean")
@ViewScoped
public class DataBean implements Serializable  {

    public DataBean() {
    }

    private static final long serialVersionUID = -6996992412087723373L;

    @NotNull
    @Size(min=2,max=10)
    private String departmentName = "fieldMBean";

    @EJB
    private HrSessionLocal sessionFacade;

    @Inject
    private DataBeanCDI dataBeanCDI;

    public void setSessionFacade(HrSessionLocal sessionFacade) {
        this.sessionFacade = sessionFacade;
    }
```


QUIZ
Grant's Tomb?
A
B. Larrington
C. Holden
What is the fourth planet?
A. Mars
B. Venus
C. Uranus

1. What is wrong with the following interface?

```
public interface House {
    @Deprecated
    void open();
    void openFrontDoor();
    void openBackDoor();
}
```

```
public interface House {
    /**
     * @deprecated use of open
     * is discouraged, use
     * openFrontDoor or
     * openBackDoor instead.
     */
    @Deprecated
    public void open();
    public void openFrontDoor();
    public void openBackDoor();
}
```

2. Consider this implementation of the `House` interface, shown in Question 1.

```
public class MyHouse implements House {
    public void open() {}
    public void openFrontDoor() {}
    public void openBackDoor() {}
}
```

```
public class MyHouse implements House {
    @SuppressWarnings("deprecation")
    public void open() {}
    public void openFrontDoor() {}
    public void openBackDoor() {}
}
```

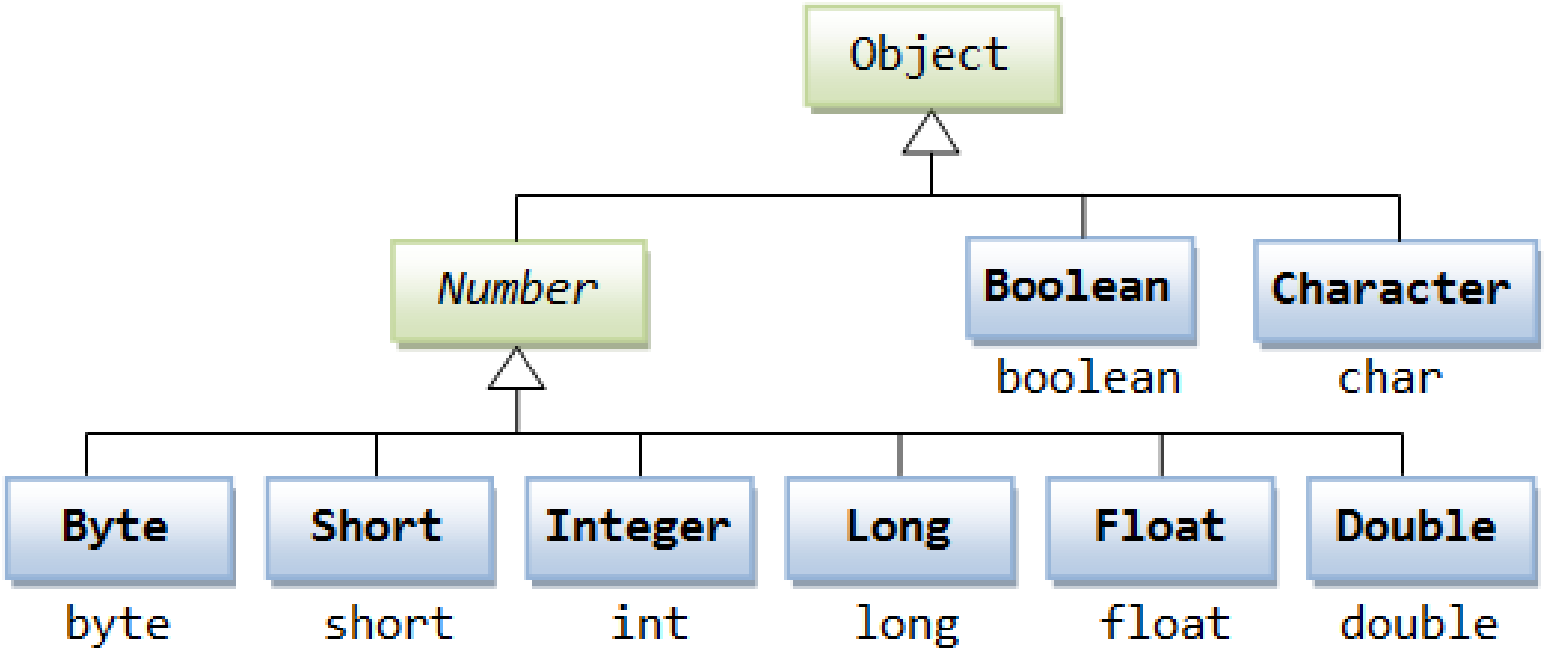
If you compile this program, the compiler produces a warning because open was deprecated (in the interface). What can you do to get rid of that warning?

Will the following code compile without error? Why or why not?

```
public @interface Meal { ... }

@Meal("breakfast", mainDish="cereal")
@Meal("lunch", mainDish="pizza")
@Meal("dinner", mainDish="salad")
public void evaluateDiet() { ... }
```

```
@java.lang.annotation.Repeatable(MealContainer.class)
public @interface Meal { ... }

public @interface MealContainer {
    Meal[] value();
}
```

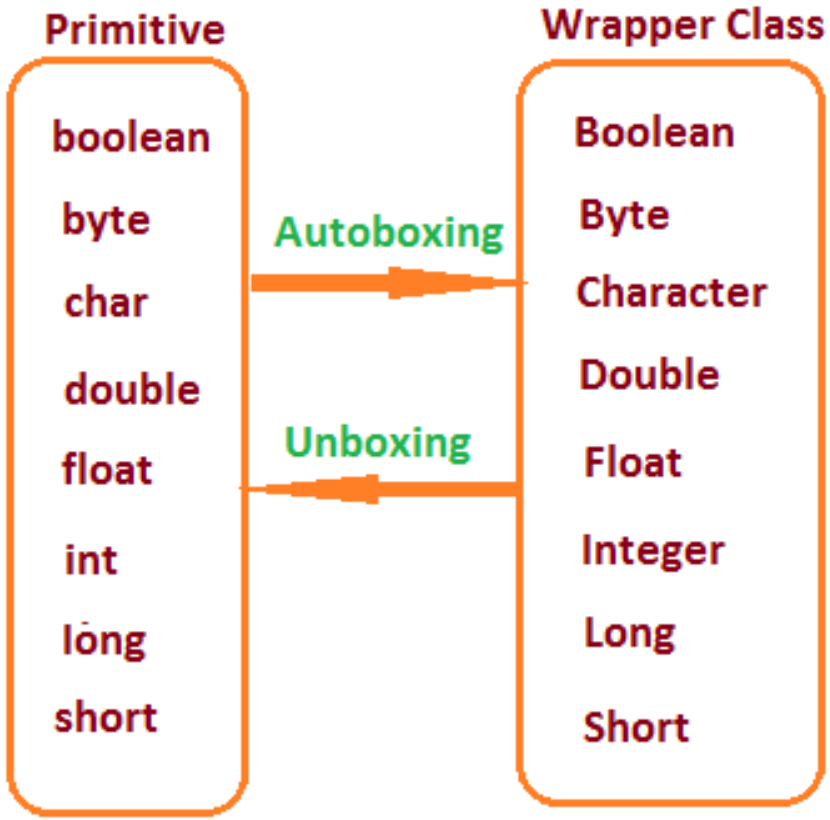
Wrapper Classes



Wrapper Classes

- Wrapper Classes have a lot of methods in common :
  - toString() Method :
    - For Example :
      - String s = Integer.toString(5);
      - String s = Character.toString('a');
  - parse Method : Converts String to an Int , float, double ,..
    - Int x = Integer.parseInt("1234");
    - double x = Double.parseDouble("12.1545");
  - Minimum and Maximum Values of a Primitive type
    - Int min = Integer.MIN_VALUE; //min =-2147483648
    - Int max = Integer.MAX_VALUE; // max = 2147483647
    - float maxv = Float.MAX_VALUE; //maxv = 3.4028235E38

Autoboxing and Unboxing




```
class Main {
    public static void main(String [] args) {
        // Boxing
        Integer a = 2;

        // UnBoxing
        int s = 5 + a;
    }
}
```

Type Safety

- The **Java** language is designed to enforce **type safety**. This means that programs are prevented from accessing memory in inappropriate ways. More specifically, every piece of memory is part of some**Java** object.

- "**Type safe**" in Java ensure that an operation is working on the right kind of data at some point before the operation is actually performed. This may be at compile time or at run time.

Java Collections

Java Collections

- A *collection* — sometimes called a container — is simply an object that groups multiple elements into a single unit.

- Collections are used to store, retrieve, manipulate, and communicate aggregate data.

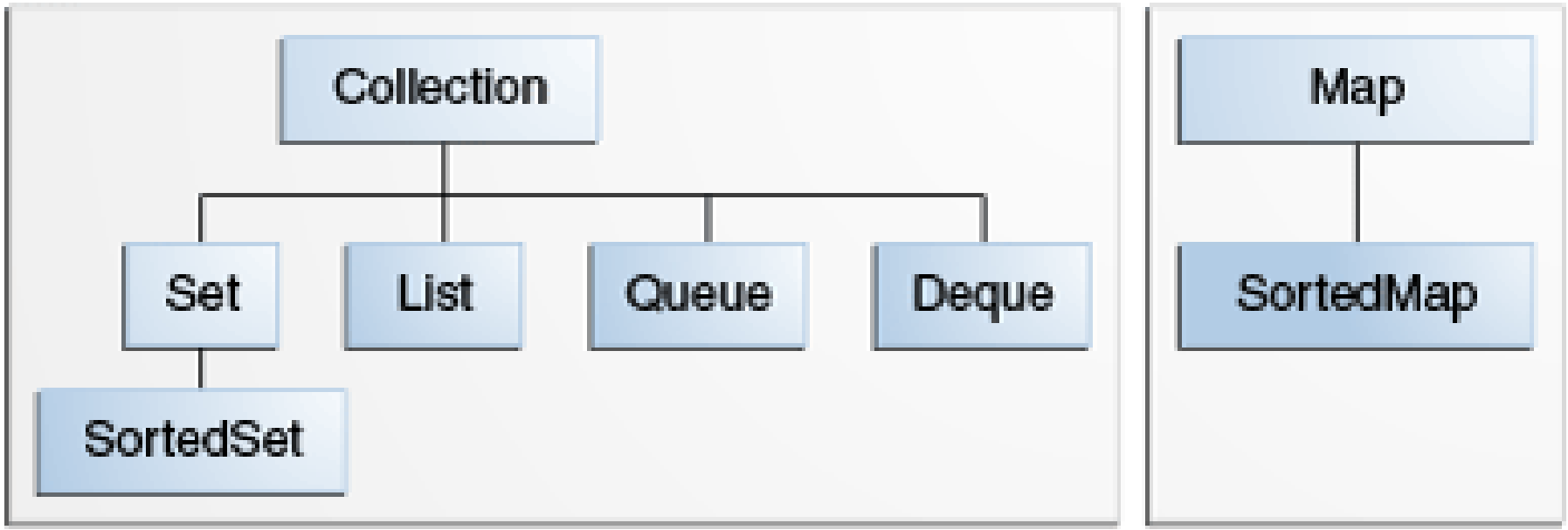
Java Collection Framework

- A *collections framework* is a unified architecture for representing and manipulating collections

- Interfaces
- Implementations
- Algorithms

- Reduces programming effort
- Increases program speed and quality

Java Collection Interfaces



The core collection interfaces.


QUIZ

Question: At the beginning of this lesson, you learned that the core collection interfaces are organized into two distinct inheritance trees. One interface in particular is not considered to be a true Collection, and therefore sits at the top of its own tree. What is the name of this interface?

Question: What interface represents a collection that does not allow duplicate elements?

Question: What interface forms the root of the collections hierarchy?

Question: What interface represents an ordered collection that may contain duplicate elements?

Question: What interface represents a collection that holds elements prior to processing?

Question: What interface represents a type that maps keys to values?

Question: What interface represents a double-ended queue?

Exercise: Consider the four core interfaces, Set, List, Queue, and Map.

- Whimsical Toys Inc (WTI) needs to record the names of all its employees. Every month, an employee will be chosen at random from these records to receive a free toy.

Exercise: Consider the four core interfaces, Set, List, Queue, and Map.

- WTI has decided that each new product will be named after an employee — but only first names will be used, and each name will be used only once. Prepare a list of unique first names.

Exercise: Consider the four core interfaces, Set, List, Queue, and Map.

- WTI decides that it only wants to use the most popular names for its toys. Count up the number of employees who have each first name.

Exercise: Consider the four core interfaces, Set, List, Queue, and Map.

- WTI acquires season tickets for the local football team, to be shared by employees. Create a waiting list for this popular sport.