Interfaces

Exercises and solutions

1. What are interfaces in Java? What is a marker interface? What is a functional interface?

**Answer:**

An interface in Java defines a reference type to specify an abstract concept. It is a specification that is meant to be implemented by classes.

An interface which has no declared or inherited members is known as a marker or tag interface.

An interface with just one abstract method is known as a functional interface.

1. What keyword do you use to implement an interface to class?

**Answer:**

implements

1. How many interfaces can a class implement?

**Answer:**

A class can implement multiple interfaces.

1. What keyword do you use in an interface declaration to inherit the interface from other interfaces?

**Answer:**

extends

1. Can you declare instance variables in an interface?

**Answer:**

No. An interface cannot have instance variables.

1. Which version of Java SE allows you have to have private methods in an interface?

**Answer:**

Java SE 9

1. What kinds of methods in an interface can be declared private? Can you have an abstract private method in an interface? If no, explain your answer.

**Answer:**

A private method in an interface can be a non-abstract, non-default instance method, or a static method.

Abstract private method in an interface does not make sense. A private method is not inherited, so it cannot be overridden, whereas an abstract method must be overridden to be useful.

1. What interface do you implement in a class to implement natural sorting for the objects of the class? What interface do you use to implement custom sorting for the objects of the class?

**Answer:**

Implement the Comparable interface for natural sorting. Implement the Comparator interface for custom sorting.

1. Describe the reason why the following interface declaration does not compile and suggest a fix.  
     
   public interface Choices {  
    int YES;  
    int NO = 1;  
    private int CANCEL = 2;  
   }

**Answer:**

The compile-time errors can be fixed as follows:

* The implicitly public static and final variable YES is not initialized. This can be fixed by assigning a value (say 0) to YES.
* The CANCEL field cannot be declared as private because all fields in an interface are implicitly public, static and final. This can be fixed by removing the keyword private.

1. What is wrong with the following interface declaration?  
     
   public interface ScheduledJob {  
    public void run() {  
    System.out.println("Running the job...");   
    }  
   }

**Answer:**

The abstract method run() cannot have a body.

1. Consider the following declaration of an interface named Greeting.  
     
   interface Greeting {  
    void sayHello();  
   }  
     
   Create a class named Greeter that implements the Greeting interface in such a way that when the following snippet of code is executed, it should print "Hello" on the standard output.  
     
   Greeting g = new Greeter();  
   g.sayHello();

**Solution:**

public class Greeter implements Greeting {

@Override

public void sayHello() {

System.out.println("Hello");

}

}

1. The following interface declaration does not compile. Describe the reason and suggest a fix.  
     
   public final interface Colorable {  
    public void color();  
   }

**Answer:**

An interface is a specification that is meant to be implemented by classes. The keyword final in the declaration indicates that the interface cannot be extended. This is the reason that you get a compile-time error. This can be fixed by removing the final modifier from the declaration.

1. Is the following interface declaration valid? What is the special name for an interface like the Sensitive interface?  
     
   public interface Sensitive {  
    // No code goes here  
   }

**Answer:**

Yes, The Sensitive interface declaration is valid. It’s a marker interface.

1. Will the following interface declaration compile? If no, give the reason.  
     
   @FunctionalInterface  
   public interface Runner {  
    public void run();  
   }

**Answer:**

Yes. The Runner interface is a functional interface and it has exactly one abstract method.

1. Is the following declaration for the Printer interface a valid a functional interface declaration? Describe your reasons how it fits or does not fit the definition of a functional interface.  
     
   @FunctionalInterface  
   public interface Printer {  
    public void print();  
     
    public default void sayHello() {  
    System.out.println("Hello");  
    }  
   }

**Answer:**

Yes, The Printer interface is a valid functional interface. It contains exactly one abstract method.

1. Consider the following declarations.  
     
   public interface Greeting {  
    default void greet() {  
    System.out.println("Hello");  
    }  
   }  
     
   public class EnglishGreeting implements Greeting {  
   }  
     
   public class HispanicGreeting implements Greeting {  
    @Override  
    public void greet() {  
    System.out.println("Ola");  
    }  
   }  
     
   what will be the output when the following snippet of code is executed?  
     
   Greeting usGreeting = new EnglishGreeting();  
   Greeting mxGreeting = new HispanicGreeting();  
   usGreeting.greet();  
   mxGreeting.greet();

**Answer:**

Hello

Ola

1. Consider the following partial declaration of an Item class.  
     
   public class Item implements Comparable<Item> {  
    private String name;  
    private double price;  
    /\* Your code goes here \*/   
   }  
     
   Complete the Item class by adding needed constructor to allow for initial value for the name and price of the item. Also add getters and setters for the two instance variables. Add the required method, so the class implements the Comparable<Item> interface. The natural order for sorting items is by their names.

**Solution:**

public class Item implements Comparable<Item> {

private String name;

private double price;

public Item(String name, double price) {

this.name = name;

this.price = price;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

@Override

public int compareTo(Item o) {

// Assuming that name is required for an item

return this.name.compareTo(o.name);

}

}

1. Create a custom comparator class – a class that implements the Comparator<Item> interface. The comparator class will sort the objects of the Item class by price and then by name.

**Solution:**

public class ItemComparator implements Comparator<Item> {

@Override

public int compare(Item i1, Item i2) {

int diff = Double.compare(i1.getPrice(), i2.getPrice());

if (diff == 0) {

diff = i1.getName().compareTo(i2.getName());

}

return diff;

}

}

You can also use the comparing() and thenComparing() methods of the Comparator interface to create a Comparator<Item> that will sort items based on price and then name as follows. In this case, you do not need to create a custom class that implements the Comparator<Item> interface as in the previous solution,

Comparator<Item> itemComparatorObject = Comparator.comparing(Item::getPrice)

.thenComparing(Item::getName);

1. Consider the following declarations for the Greeting interface and the Greeter class:  
     
   public interface Greeting {  
    default void greet() {  
    System.out.println("Namaste");  
    }  
   }  
     
   public class Greeter implements Greeting {   
    @Override  
    public void greet() {  
    /\* Calls the greet() method of the Greeting interface here \*/  
     
    System.out.println("Hello");  
    }  
   }  
     
   Complete the code in the greet() method of the Greeter class by adding one statement as the first statement in the method. The statement should call the greet() method of the Greeting interface. When the following snippet of code is executed, it should print "Namaste" and "Hello" – each word in a separate line.  
     
   Greeting g = new Greeter();  
   g.greet();   
     
   The expected output is as follows:  
   Namaste  
   Hello

**Solution:**

public interface Greeting {

default void greet() {

System.out.println("Namaste");

}

}

public class Greeter implements Greeting {

@Override

public void greet() {

**Greeting.super.greet();**

System.out.println("Hello");

}

}