Object Oriented Programming with Java 8

Akshita Chanchlani

Interface

Interface

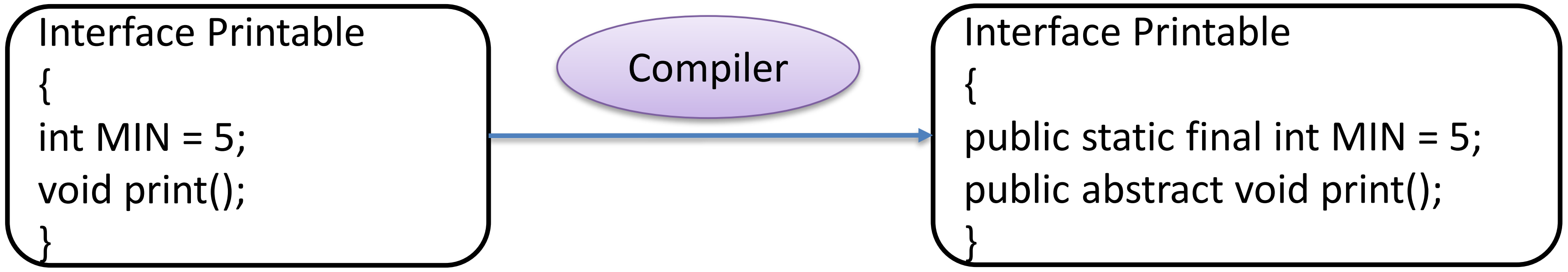
- In Java, an **interface** is a blueprint or template of a class. It is much similar to the Java class but the only difference is that it has abstract methods and static constants.
- An interface provides specifications of what a class should do or not and how it should do. An interface in Java basically has a set of methods that class may or may not apply.
- It also has capabilities to perform a function. The methods in interfaces do not contain any body.
- An interface in Java is a mechanism which we mainly use to achieve abstraction and multiple inheritances in Java.
- An interface provides a set of specifications that other classes must implement.
- We can implement multiple Java Interfaces by a Java class. All methods of an interface are implicitly public and abstract. The word abstract means these methods have no method body, only method signature.
- Java Interface also represents the IS-A relationship of inheritance between two classes.
- An interface can inherit or extend multiple interfaces.
- We can implement more than one interface in our class.

Interface Vs Class

- Unlike a class, you cannot instantiate or create an object of an interface.
- All the methods in an interface should be declared as abstract.
- An interface does not contain any constructors, but a class can.
- An interface cannot contain instance fields. It can only contain the fields that are declared as both static and final.
- An interface can not be extended or inherited by a class; it is implemented by a class.
- An interface cannot implement any class or another interface.

Syntax Interface

```
interface interface-name
{
//abstract methods
}
```



```
Interface Printable
{
int MIN = 5;
void print();
}
```

```
Interface Printable
{
public static final int MIN = 5;
public abstract void print();
}
```

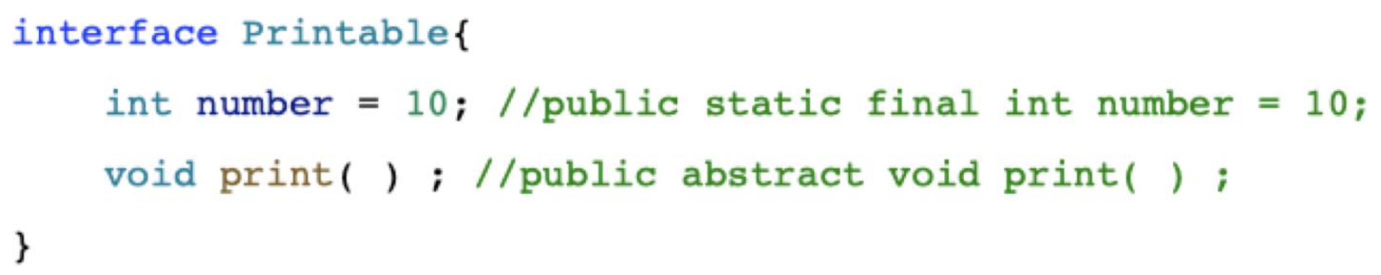
Interface

- Set of rules are called specification/standard.
- It is a contract between service consumer and service provider.
- If we want to define specification for the sub classes then we should define interface.
- Interface is non primitive type which helps developer:
  1. To build/develop trust between service provider and service consumer.
  2. To minimize vendor dependency.
- interface is a keyword in Java.

```
interface Printable{
    //TODO
}
```

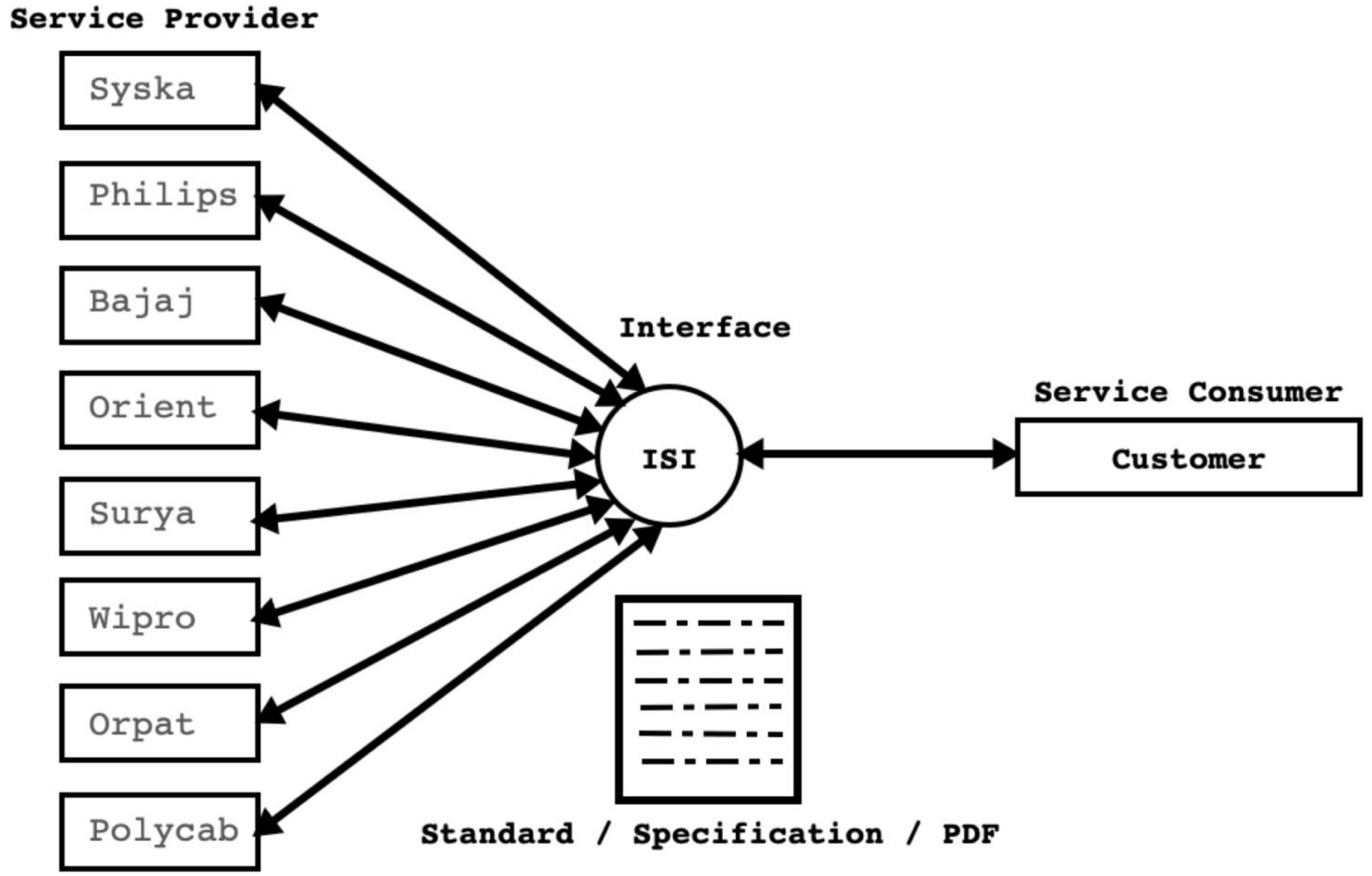
Interface

- Interface can contain:
    1. Nested interface
    2. Field
    3. Abstract method
    4. Default method
    5. Static method
- Interfaces cannot have constructors.
- We can create reference of interface but we can not create instance of interface.
- We can declare fields inside interface. Interface fields are by default public static and final.
- We can write methods inside interface. Interface methods are by default considered as public and abstract.



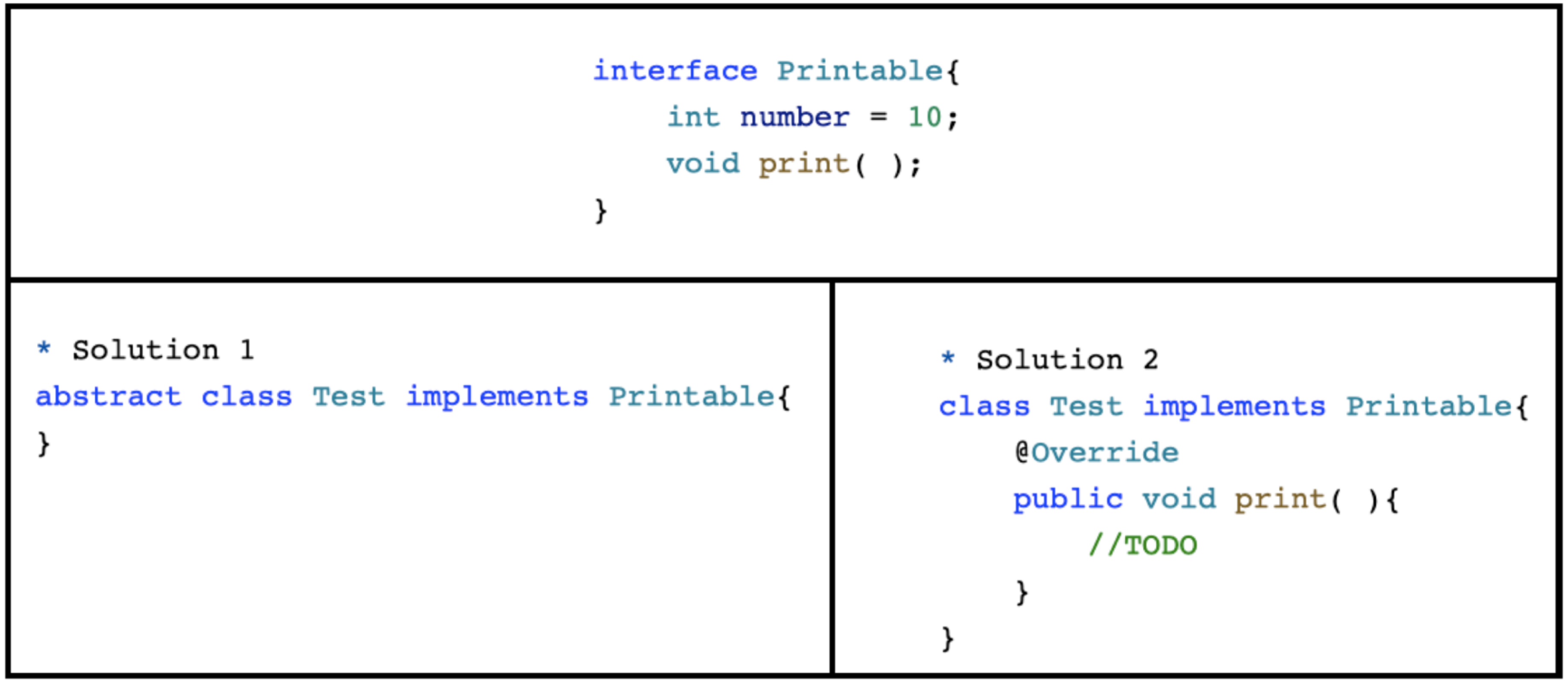
```
interface Printable{
    int number = 10; //public static final int number = 10;
    void print( ) ; //public abstract void print( ) ;
}
```

Interface


Service Provider
Syska
Philips
Bajaj
Orient
Surya
Wipro
Orpat
Polycab
Interface
ISI
Service Consumer
Customer
Standard / Specification / PDF

Interface

- If we want to implement rules of interface then we should use implements keyword.
- It is mandatory to override, all the abstract methods of interface otherwise sub class can be considered as abstract.

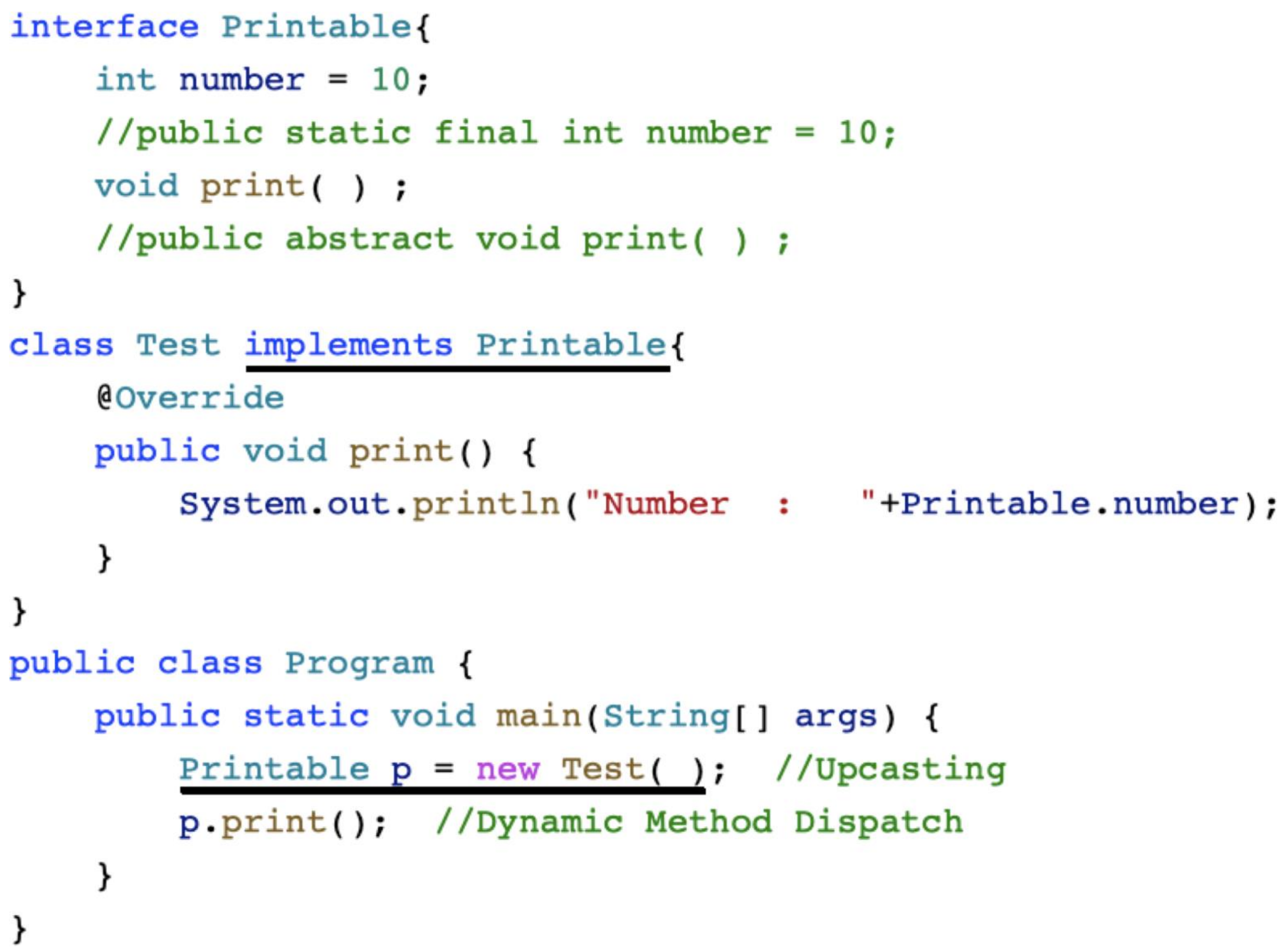


```
interface Printable{
    int number = 10;
    void print( );
}
```

```
* Solution 1
abstract class Test implements Printable{
}
```

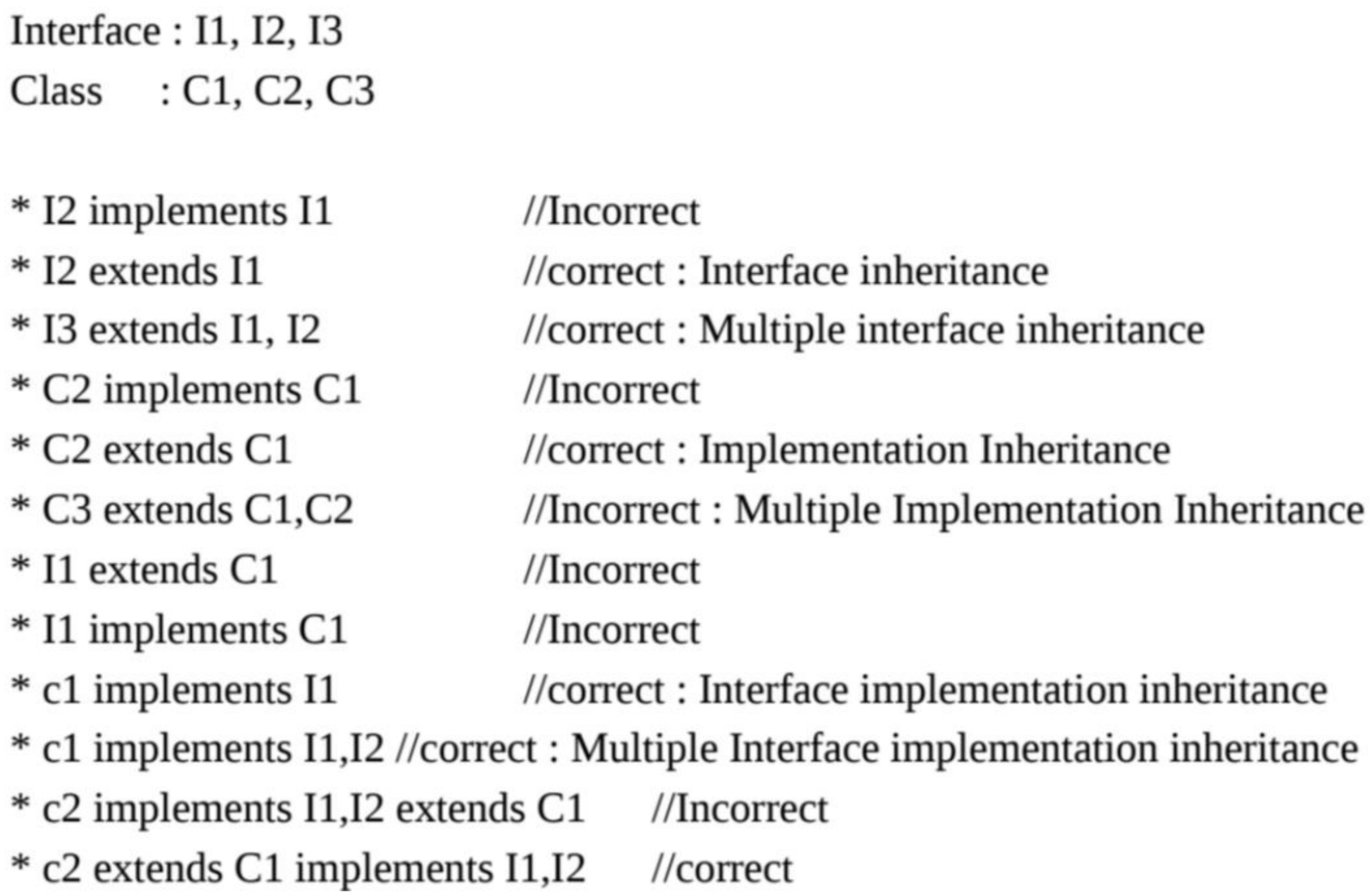
```
* Solution 2
class Test implements Printable{
    @Override
    public void print( ){
        //TODO
    }
}
```

Interface Implementation Inheritance



```java
interface Printable{
    int number = 10;
    //public static final int number = 10;
    void print( ) ;
    //public abstract void print( ) ;
}
class Test implements Printable{
    @Override
    public void print() {
        System.out.println("Number  :   "+Printable.number);
    }
}
public class Program {
    public static void main(String[] args) {
        Printable p = new Test( );  //Upcasting
        p.print();  //Dynamic Method Dispatch
    }
}
```

Interface Syntax



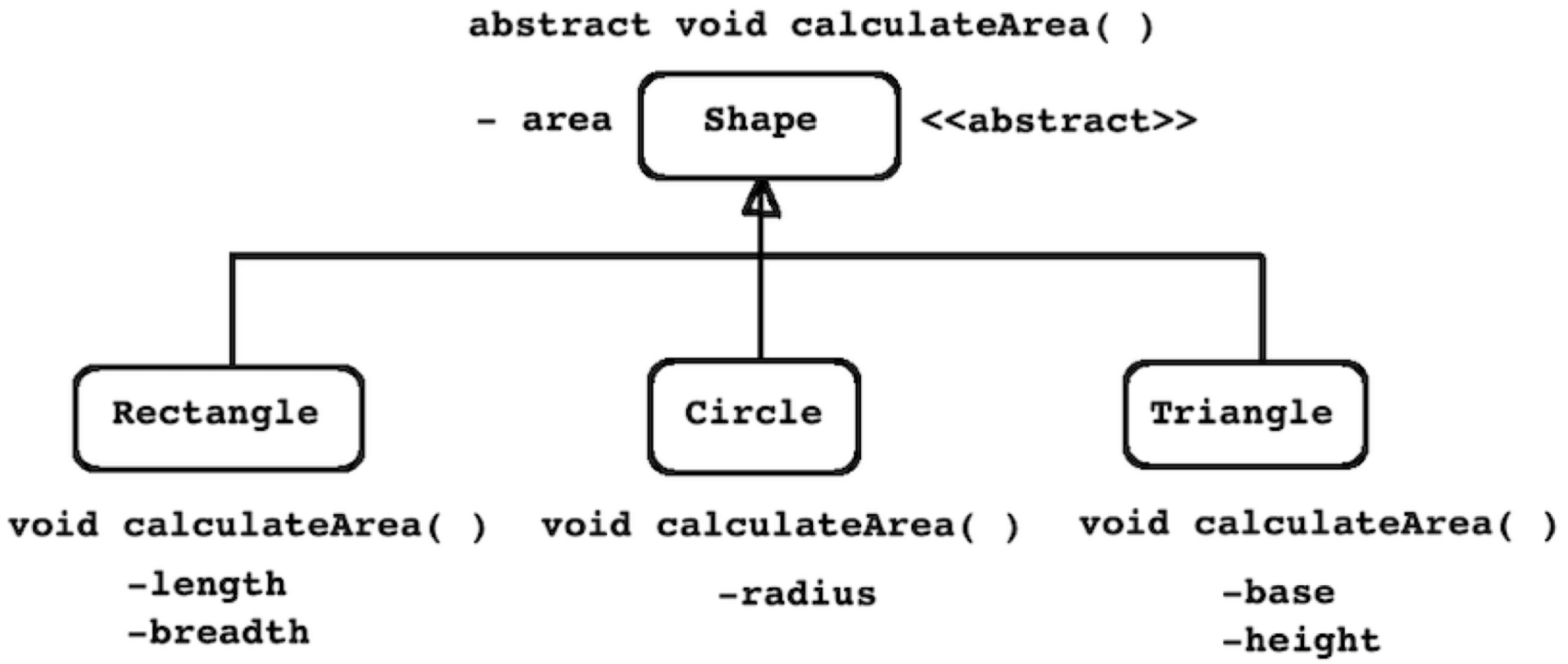
```
Interface : I1, I2, I3
Class     : C1, C2, C3

* I2 implements I1              //Incorrect
* I2 extends I1                 //correct : Interface inheritance
* I3 extends I1, I2             //correct : Multiple interface inheritance
* C2 implements C1              //Incorrect
* C2 extends C1                 //correct : Implementation Inheritance
* C3 extends C1,C2              //Incorrect : Multiple Implementation Inheritance
* I1 extends C1                 //Incorrect
* I1 implements C1              //Incorrect
* c1 implements I1              //correct : Interface implementation inheritance
* c1 implements I1,I2 //correct : Multiple Interface implementation inheritance
* c2 implements I1,I2 extends C1      //Incorrect
* c2 extends C1 implements I1,I2      //correct
```

Types of inheritance

- **Interface Inheritance**
  - During inheritance if super type and sub type is interface then it is called interface inheritance.
    1. Single Inheritance(Valid in Java)
    2. Multiple Inheritance(Valid in Java)
    3. Hierarchical Inheritance(Valid in Java)
    4. Multilevel Inheritance(Valid in Java)
- **Implementation Inheritance**
  - During inheritance if super type and sub type is class then it is called implementation inheritance.
    1. Single Inheritance(Valid in Java)
    2. Multiple Inheritance(Invalid in Java)
    3. Hierarchical Inheritance(Valid in Java)
    4. Multilevel Inheritance(Valid in Java)

Abstract Class



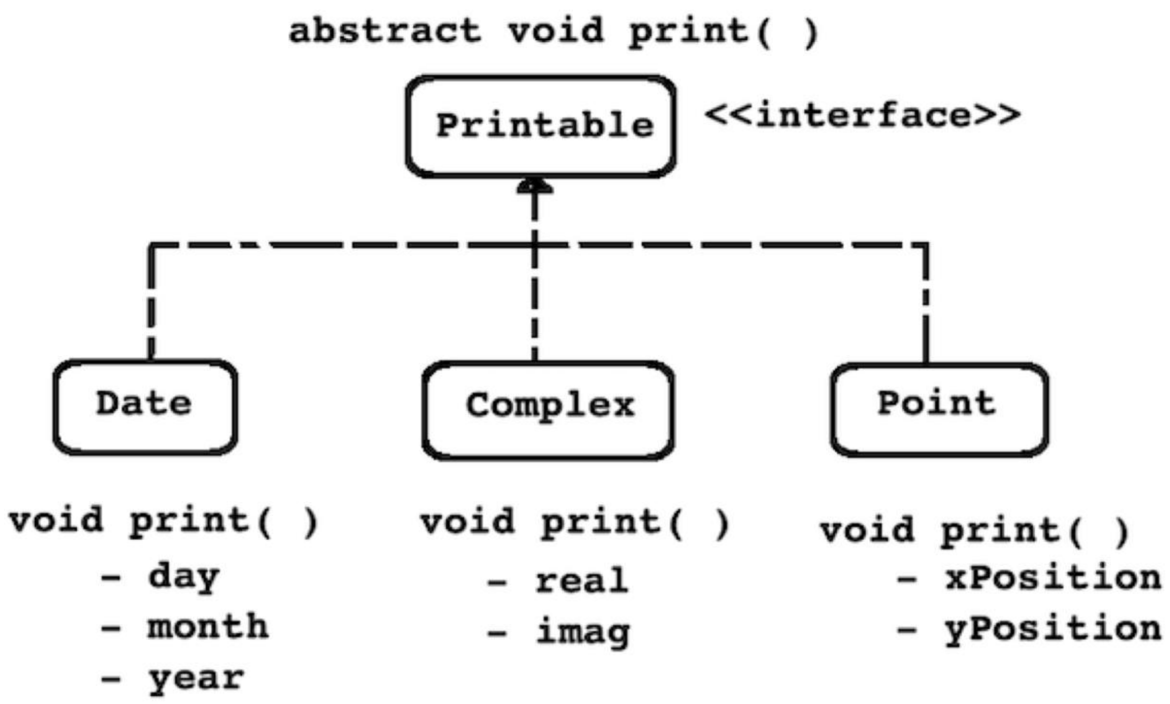
```
Shape[] arr = new Shape[ 3 ];
arr[ 0 ] = new Rectangle( ); //Upcasting
arr[ 1 ] = new Circle( ); //Upcasting
arr[ 2 ] = new Triangle( ); //Upcasting
```

Abstract Class

1. If "is-a" relationship is exist between super type and sub type and if we want same method design in all the sub types then super type must be abstract.
2. Using abstract class, we can group instances of related type together
3. Abstract class can extend only one abstract/concrete class.
4. We can define constructor inside abstract class.
5. Abstract class may or may not contain abstract method.

- **Hint** : In case of inheritance if state is involved in super type then it should be abstract.

Interface



```
Printable[] arr = new Printable[ 3 ];
arr[ 0 ] = new Date( ); //Upcasting
arr[ 1 ] = new Complex( ); //Upcasting
arr[ 2 ] = new Point( ); //Upcasting
```

Interface

1. If "is-a" relationship is not exist between super type and sub type and if we want same method design in all the sub types then super type must be interface.

2. Using interface, we can group instances of unrelated type together.

3. Interface can extend more than one interfaces.

4. We can not define constructor inside interface.

5. By default methods of interface are abstract.

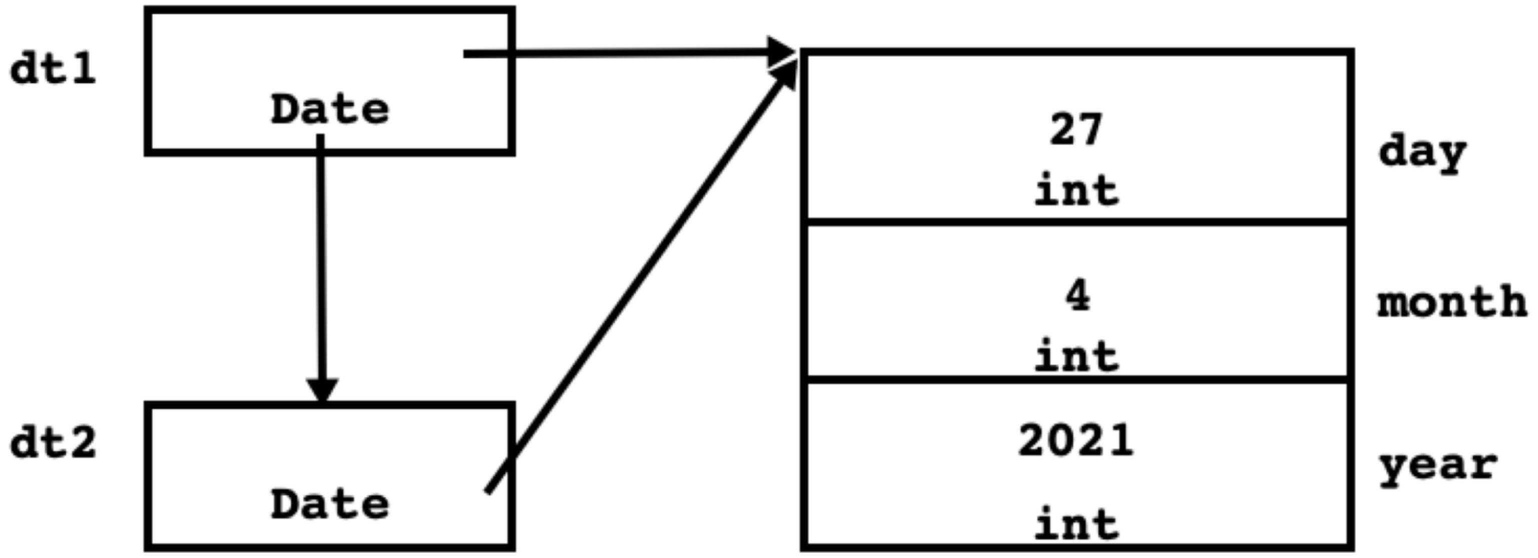
- **Hint** : In case of inheritance if state is not involved in super type then it should be interface.

Commonly Used Interfaces

1. java.lang.AutoCloseable
2. java.io.Closeable
3. java.lang.Cloneable
4. java.lang.Comparable
5. java.util.Comparator
6. java.lang.Iterable
7. java.util.Iterator
8. java.io.Serializable

Cloneable Interface Implementation

- `Date dt1 = new Date( 27, 4, 2021 );`
- `Date dt2 = dt1;  //Shallow Copy Of References`

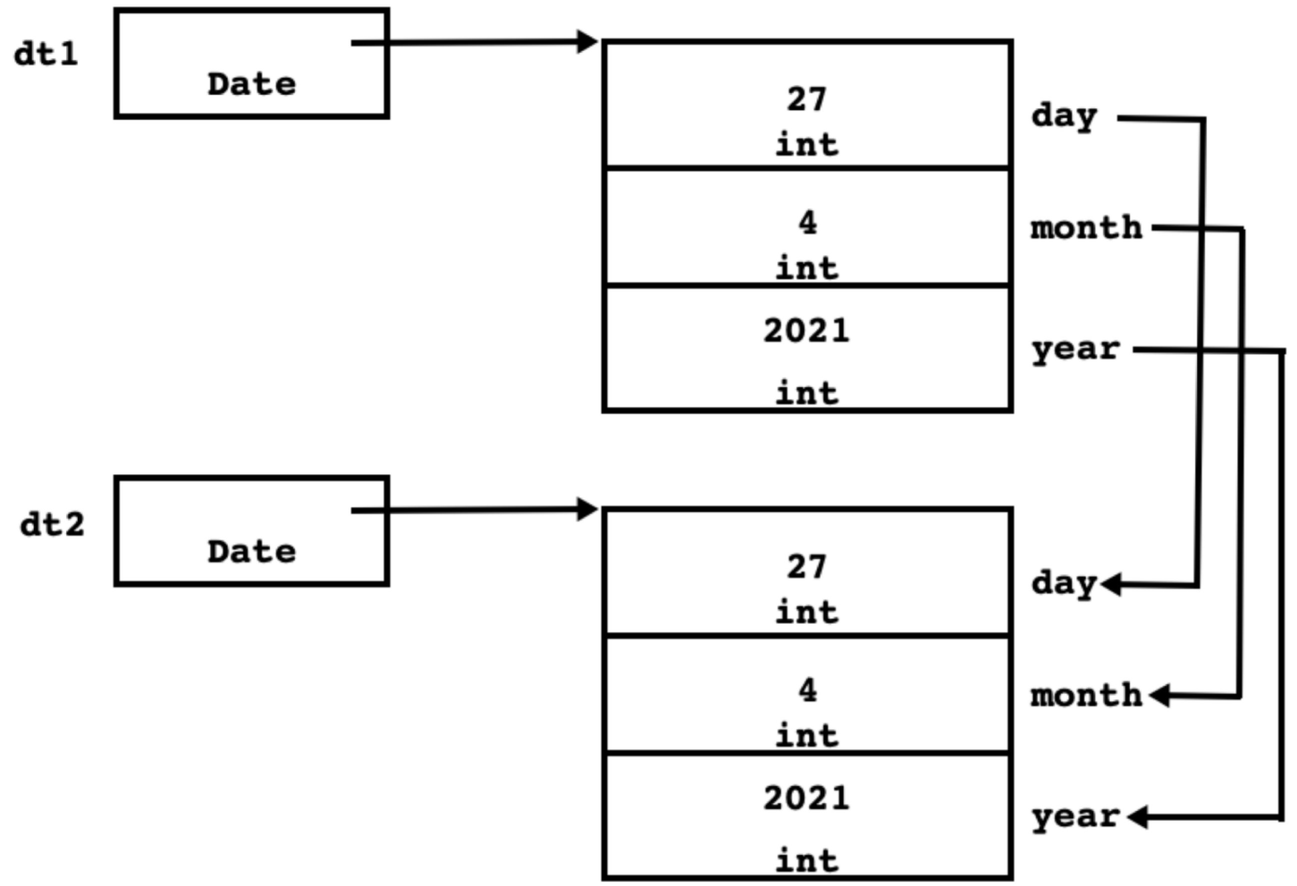


Cloneable Interface Implementation

- If we want to create new instance from existing instance then we should use clone method.
- clone( ) is non final native method of java.lang.Object class.
- Syntax:
  - **protected native Object clone( ) throws CloneNotSupportedException**
- Inside clone() method, if we want to create shallow copy instance then we should use super.clone( ) method.
- Cloneable is interface declared in java.lang package.
- Without implementing Cloneable interface, if we try to create clone of the instance then clone() method throws CloneNotSupportedException.

Cloneable Interface Implementation

- `Date dt1 = new Date( 27, 4, 2021 );`
- `Date dt2 = dt1.clone( ); //Shallow Copy Of Instance`



Marker Interface

- An interface which do not contain any member is called marker interface. In other words, empty interface is called as marker interface.
- Marker interface is also called as tagging interface.
- If we implement marker interface then Java compiler generates metadata for the JVM, which help JVM to clone/serialize or marshal state of object.
- Example:
    1. java.lang.Cloneable
    2. java.util.EventListener
    3. java.util.RandomAccess
    4. java.io.Serializable
    5. java.rmi.Remote

Comparable

- It is interface declared in java.lang package.
- **"int compareTo(T other)"** is a method of java.lang.Comparable interface.
- If state of current object is less than state of other object then compareTo() method should return negative integer(-1).
- If state of current object is greater than state of other object then compareTo() method should return positive integer(+1).
- If state of current object is equal to state of other object then compareTo() method should return zero(0).
- If we want to sort, array of non primitive type which contains all the instances of same type then we should implement Comparable interface.

Comparator

- It is interface declared in java.util package.
- **"int compare(T o1, T o2)"** is a method of java.util.Comparator interface.
- If state of current object is less than state of other object then compare() method should return negative integer(-1).
- If state of current object is greater than state of other object then compare() method should return positive integer(+1).
- If state of current object is equal to state of other object then compare() method should return zero(0).
- If we want to sort, array of instances of non primitive of different type then we should implement Comparator interface.

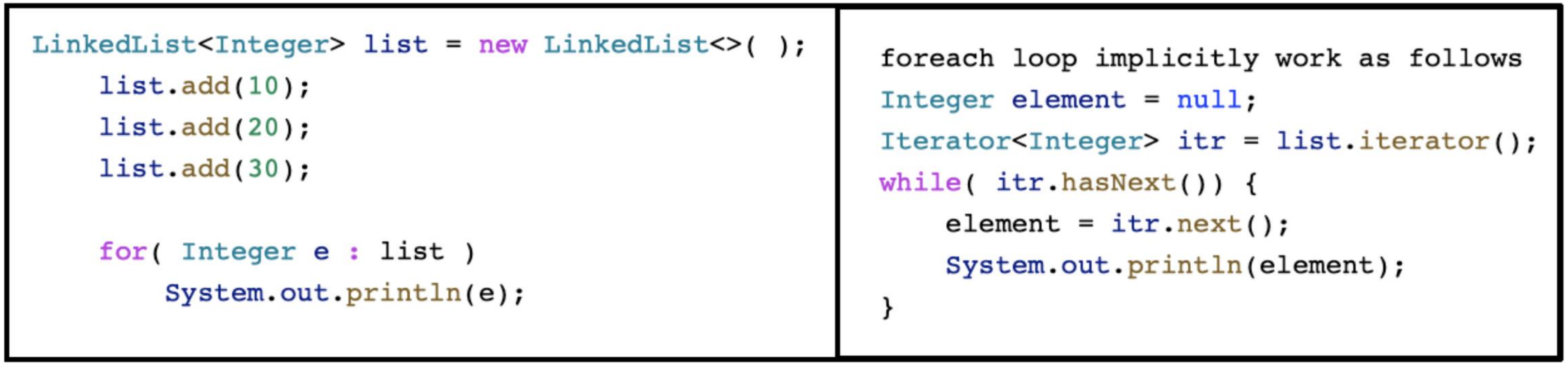
Iterable and Iterator Implementation

- Iterable<T> is interface declared in java.lang package.
- Implementing this interface allows an object to be the target of the "for-each loop" statement.
- It is introduced in JDK 1.5
- Methods of java.lang.Iterable interface:
    1. Iterator<T> iterator()
    2. default Spliterator<T> spliterator()
    3. default void forEach(Consumer<? super T> action)

Iterable and Iterator Implementation

- Iterator<E> is interface declared in java.util package.
- It is used to traverse collection in forward direction only.
- It is introduced in JDK 1.2
- Methods of java.util.Iterator interface:
    1. boolean hasNext()
    2. E next()
    3. default void remove()
    4. default void forEachRemaining(Consumer<? super E> action)

Iterable and Iterator Implementation



```
LinkedList<Integer> list = new LinkedList<>( );
    list.add(10);
    list.add(20);
    list.add(30);

    for( Integer e : list )
        System.out.println(e);
```

```
foreach loop implicitly work as follows
Integer element = null;
Iterator<Integer> itr = list.iterator();
while( itr.hasNext()) {
    element = itr.next();
    System.out.println(element);
}
```

Thank You.

akshita.chanchlani@sunbeaminfo.com