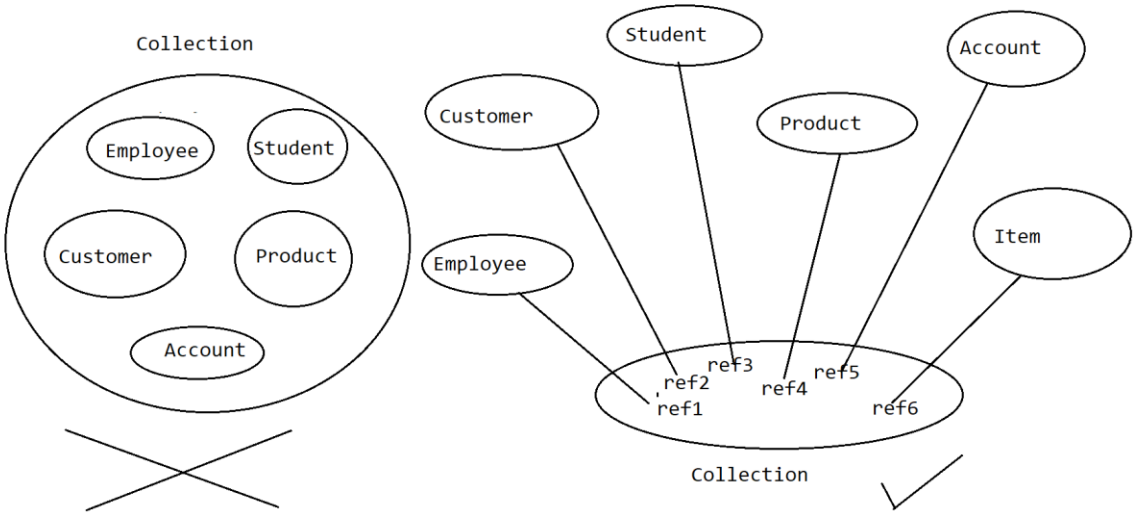
Wrapper Classes:
—---------------
Collection: It is an object, it is able to store a group of other objects.

Collection objects are not providing space for the Objects, Collection objects are able to store reference values of the other objects.



IN Java applications, Collection objects are able to store Object reference values only, not primitive data.

As per the requirement , if we want to store primitive data in Collection objects then we have to use the following steps.

1. Declare data in the primitive variables
2. Convert data from Primitive variable to an Object.
3. Store the generated object in the Collection Object.

If we want to read primitive data from Collection objects then we have to use the following steps.

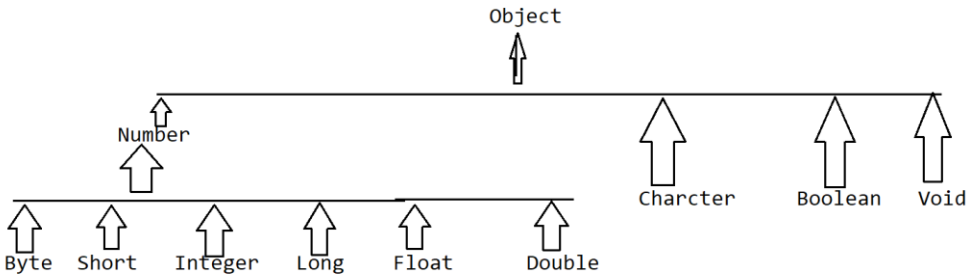
1. Read Object from the Collection Object.
2. Convert data from the Object to a primitive variable.

In the above context, to convert data from primitive type to Object type and from Object type to primitive type Java has provided a set of predefined classes called “Wrapper Classes”.

Java has provided all the wrapper classes in java.lang package as Immutable classes.

Java has provided a separate wrapper class for each and every primitive data type.

byte----------> java.lang.Byte
short----------> java.lang.Short
int----------> java.lang.Integer
long----------> java.lang.Long
float----------> java.lang.Float
double----------> java.lang.Double
char----------> java.lang.Character
boolean----------> java.lang.Boolean



Note: Java has provided all the wrapper classes as implementation classes to java.io.Serializable marker interface.

In Java applications, by using wrapper classes we are able to perform the following conversions.

Conversions from Primitive Type to Object type:
---

1. By Using Parameterized constructor from wrapper classes:
   public XXX(xxx val)
   Where XXX is Wrapper class.
   Where xxx is a primitive data type.
   EX: public Integer(int val)
   EX: public Float(float val)

EX:

```
public class Main {
    public static void main(String[] args) {
        int i = 10;
```

```
        Integer in = new Integer(i);
        System.out.println(i+"    "+in);
    }
}
```

2. By Using valueOf() method from Wrapper classes:
   public static XXX valueOf(xxx val)
   Where XXX is wrapper class
   Where xxx is a primitive type

EX:

```
public class Main {
    public static void main(String[] args) {
        int i = 10;
        Integer in = Integer.valueOf(i);
        System.out.println(i+"    "+in);
    }
}
```

3. By Using Auto-Boxing Mechanism:
   It was introduced in JDK1.5 version, where we can assign primitive variables to the wrapper class reference variables directly without using any predefined constructor and method.

EX:

```
public class Main {
    public static void main(String[] args) {
        int i = 10;
        Integer in = i;
        System.out.println(i+"    "+in);
    }
}
```

Conversions from Object Type to primitive type:

---

1. By Using xxxValue() method from Wrapper classes:
   public xxx xxxValue()
   Where xxx is a primitive data type.

EX:

```
public class Main {
    public static void main(String[] args) {
```

```
        Integer in = new Integer(10);
        int i = in.intValue();
        System.out.println(in+"     "+i);
    }
}
```

2. By Using Auto Unboxing Mechanism:
   It was introduced in JDK1.5 version, where we can assign the Wrapper class reference variable to the respective primitive variable.

EX:

```
public class Main {
    public static void main(String[] args) {
        Integer in = new Integer(10);
        int i = in;
        System.out.println(in+"     "+i);
    }
}
```

Conversions from String type to Object Type:

1. By Using String parameterized constructor from Wrapper classes:
   public XXX(String val)
   Where XXX is a Wrapper class.

EX:

```
public class Main {
    public static void main(String[] args) {
        String val = "10";
        Integer in = new Integer(val);
        System.out.println(val+"    "+in);
    }
}
```

2. By Using valueOf() method from Wrapper classes:
   public static XXX valueOf(String val)

EX:

```
public class Main {
    public static void main(String[] args) {
        String val = "10";
        Integer in = Integer.valueOf(val);
```

```
        System.out.println(val+"    "+in);
    }
}
```

Conversions from Object Type to String type:

1. By using toString() method from Wrapper classes:
   public String toString()

EX:

```
public class Main {
    public static void main(String[] args) {
        Integer in = new Integer(10);
        String val = in.toString();
        System.out.println(in+"    "+val);
    }
}
```

2. By Using '+' concatenation Operator:
   If we concatenate any wrapper class object reference variable with "" by using + operator then JVM will access toString() method internally and JVM will append the return value of toString() method to the "".

EX:

```
public class Main {
    public static void main(String[] args) {
       Integer in = new Integer(10);
       String val = ""+in;// ""+in.toString();
        System.out.println(in+"   "+val);

    }
}
```

Conversions from Primitive Type to String type:

1. ByUsing static toString() method from Wrapper classes:
   public static String toString(xxx value)
   Where xxx is a primitive type.

EX:

```
public class Main {
    public static void main(String[] args) {
       int i = 10;
```

```
        String val = Integer.toString(i);
         System.out.println(i+"    "+val);

    }
}
```

2. By using '+' concatenation Operator:
   If we concatenate any primitive variable with "" by using '+' operator then the primitive value will be converted to String automatically.

EX:

```
public class Main {
    public static void main(String[] args) {
        int i = 10;
        String val = ""+i;
         System.out.println(i+"    "+val);

    }
}
```

Conversions from String type to Primitive Type:

---

1. By using parseXxx() methods from Wrapper classes.
   public xxx parseXxx(String value)

EX:

```
public class Main {
    public static void main(String[] args) {
        String val = "10";
        int i = Integer.parseInt(val);
         System.out.println(val+"    "+i);
    }
}
```

Note: In place of 10 if we provide any String value like "abc" or "xyz" then JVM will raise java.langy.NumberFormatException.

EX:

```
public class Main {
    public static void main(String[] args) {
        String val = "abc";
        int i = Integer.parseInt(val);
```

```
        System.out.println(val+"     "+i);
    }
}
```

Status: java.lang.NumberFormatException: For the input “abc”.