OBJECT ORIENTED PROGRAMMING USING

Java

Let's explore technology
together to live in the future

Checkout more on
https://github.com/Sy-hash-collab

Sy-hash-collab

Java Iterator:

An iterator is an object that can be used to loop through collections like ArrayList and Hashset.

It is called iterator bcz "iterating" is the technical term for looping.

Getting an Iterator:

The iterator() method can be used to get an Iterator for any Collection.

```
import java.util.Arraylist;
import java.util.Iterator;
public class Main{
 public static void main (String[] args)
{ Arraylist<String> cars = new Arraylist<String>()

   cars.add ("Volvo");
   cars.add ("Ford");
   cars. add ("BMW");
   cars. add ("Mazda");

 Iterator<String> it = cars.iterator();

  System.out.println (it.next());

}} Volvo
```

loop of x

prints first element in this case

"it" - object which has iteration of x

Loop through a Collection:

To loop through a collection, use the hasNext() and next() methods of Iterator:

```
import java.util.ArrayList;
import java.util.Iterator;
public class Main {
 public static void main (String [] args)
{ ArrayList <String> cars = new ArrayList<String>();

 cars.add ("Volvo");
 cars.add ("BMW");
 cars.add ("Ford");

 { Iterator<String> it = cars.iterator();

  while (it.hasNext())
 {
  System.out.println (it.next());
 }
}
}
```

```
Volvo
BMW
Ford
Mazda
```

Removing Items from a Collection:

Iterators are designed to easily change the collections that they loop through. The remove() method can remove items from a collection while looping.

```
import java.util.ArrayList;
import java.util.Iterator;
public class Main {
 public static void main (String [] args)
{ Arraylist <Integer> numbers = new Arraylist
                                  <Integer>();
 numbers.add (12);
 numbers.add (8);
 numbers.add (2);
 numbers.add (23);

Iterator <Integer> it = numbers.iterator ();

 while (it.hasNext ())
{
  Integer  i = it.next ();

  if (i < 10)
 {
  it.remove ();
 }
}
  System.out.println (numbers);
 }
}
```

Note:

Trying to remove items using a for loop or a for-each loop wouldn't work bcz collection is changing size at same tim
that code is trying to loop

```
import java.util.Iterator;
import java.util.Arraylist;
public class Main{
 public static void main (String [] args)
{ Arraylist<String> name = new ArrayList<String>();

   name.add ("ali");
   name.add ("ahmed");

Iterator<String> it = name.iterator();

  int y=0;

  while (it.hasNext());
{
 System.out.println (it.next());

  if (y==2)
   { it.remove(y);}
     y++;
}
```

Java Wrapper Class:

The wrapper classes in Java are used to convert primitive types (int, char, float etc.) into corresponding objects.

Wrapper classes provide a way to use primitive data types (int, boolean etc.) as objects.

Primitive Data Type	Wrapper Class
byte	Byte
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Sometimes, you must use wrapper classes, for example when working with Collection objects such as ArrayList, where primitive types cannot be used (the list can only store objects).

```
// Invalid
ArrayList<int> num = new ArrayList<int>();

// Valid
ArrayList<Integer> num = new ArrayList<Integer>();
```

Creating Wrapper Objects:

To create a wrapper object, use the wrapper class instead of the primitive type. To get the value, you can just print the object:

```
public class Main {
public static void main (String[]args)
{
  Integer myInt = 5;
  Double myDouble = 5.99;
  Charachter myChar = 'A';
  System.out.println (myInt);
  System.out.println (myDouble);
  System.out.println (myChar);
}
}
```

```
5
5.99
A
```

Since you're now working with objects, you can use certain methods to get information about specific object. For example, the following methods are used to get the value associated with corresponding wrapper object:

```
intValue(),
byteValue()
shortValue()
longValue()
floatValue()
doubleValue()
charValue()
```

```
public class Main{
 public static void main (String [] args)
{  Integer  myInt = 5;
   Double  myDouble = 5.99;
   Charachter  myChar = "A";

   System.out.println (myInt.intValue());
   System.out.println (myDouble.doubleValue());
   System.out.println (myChar.charValue());

}}
```

```
5
5.99
A
```

Another useful method is toString() which is used to convert wrapper objects to strings. In the following example, we convert an Integer to a String, and use the length() method of String class to output the length of "string".

```
public class Main {
 public static void main (String [] args)
{
  Integer  myInt = 100;

  String  myString = myInt.toString();

  System.out.println (myString.length());

}}
```

```
3
```

Autoboxing:

The automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing.

e.g conversion of int to Integer, long to Long.

We can use valueOf() method to convert primitive types into corresponding objects.

```
class Main {
public static void main (String [Jargs)
{  //create primitive types
   int a = 5;
   double b = 5.65;

   //converts into wrapper objects
   Integer aObj = Integer.valueOf(a);

   Double bObj = Double.valueOf(b);

  if (aObj instanceof Integer)
  { System.out.println("An object of
                        Integer is created");
  }

  if (bObj instanceof Double)
  { System.out.println("An object of
                        Double is created");
}}
```

In previous example, we have used the valueOf() method to convert the primitive types into objects.

Hence, we have used the instanceof operator to check whether the generated objects are of Integer or Double type or not.

However, the Java compiler can directly convert the primitive types into corresponding objects.

```
int a = 5;

Integer aObj = a;
```

- **Example:**

```
import java.util.ArrayList;
class Autoboxing{
 public static void main (String [] args)
{  char ch = 'a';

  Charachter c = ch;

ArrayList <Charachter> a = new ArrayList
                          <Character> ();

~~arraylist~~ a.add (c);

System.out.println (a.get (0));
}}
```

Unboxing:

It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing.

Conversion of Integer to int, Long to long

To convert objects into primitive types, we can use corresponding value methods intValue(), doubleValue() present in each wrapper class.

```
class Main{
 public  static void main (String [Jargs)
{
 //create objects of wrapper class
 Integer aObj = Integer.valueOf (23);

 Double  bObj = Double.valueOf (5.55);

 // converts into primitive types
 int  a  = aObj.intValue ();

 double b = bObj.doubleValue ();


 System.out.println ("The value of a: " + a );

 System.out.println ("The value of b: " + b);
}}
```

In previous example, we have used the intValue(), doubleValue() method to convert the Integer and Double objects into corresponding primitive types.
However, the Java compiler can automatically convert objects into corresponding primitive types.

```
Integer aObj = Integer.valueOf(2);
converts into int type
int a = aObj;
```

- **Example:**

```
import java.util.ArrayList;

class Unboxing {
public static void main(String [Jargs)
{ Character ch = "a";

  char c = ch;

ArrayList<Character> arraylist = new
                    ArrayList<Character>();

arrayList.add(c);

System.out.println(arraylist.get(0));

}
}
```

a

Need of Wrapper Classes:

1) The convert primitive data types into objects. Objects are needed if we wish to modify values / arguments passed into a method.
This means that when you pass a primitive datatype to a method in Java, you're passing a copy of the value, not the actual variable. Therefore if you want to modify the original value outside the method, you need to use objects.

Example 1:

```
public class ObjectModificationExample
{ public static void main (String[]args)
{ int primitiveInt = 40;
System.out.println ("Original primitiveInt
                    before method call");
                    + primitiveInt

modifyPrimitiveValue (primitiveInt);

System.out.println ("Original primitiveInt
                    after method call:"+
                    primitiveInt);

}
```

```
public static void modifyPrimitiveValue(int
                                        value)
{
    value = value + 10;
    System.out.println("Modified value inside
                        method: " + value);
}}
```

```
Original primitiveInt before method call : 40
Modified value inside method : 50
Original primitiveInt after method call : 40
```

Example 2 :

```
public class WrapperExample{
  public static void main(String[]args)
{
    Integer wrapper = new Integer(40);
    System.out.println("Original value before
                        method call: " + wrapper);
    modifyObjectValue(wrapper);
    System.out.println("Original value after
                        method call: " + wrapper);
}
  public static void modifyObjectValue
                        (Integer wrapper)
{
    wrapper = wrapper + 10;
    System.out.println("Modified: " + wrapper);
```

Advantages of Wrapper Classes:

- Collections allowed only object data.
- On object data we can call multiple methods compareTo(), equals(), toString()

Example 1 : compareTo()

```
public class CompareToExample {
  public static void main (String [] args)
{   Integer x = 10;
    Integer y = 20;

  int result = x.compareTo(y);

  if (result <0 )
    { System.out.println (x+ "is less than:"+y);
    }
 else if (result >0 )
 { System.out.println ( x+"is greater than:"+y);
 }
 else
 { System.out.println (x+"is equal to"+y);
}}}
```

Example 2 : equals()

```
public class EqualsExample {
 public static void main (String [] args)
{  Integer  a = 10;
   Integer  b = 10;
   Integer  c = 20;
   boolean  result = a.equals(b);
   boolean  result = a.equals (c);

  System.out.println (a + "equals "+b+results);

 System.out.println (a +"equals "+ c +": "+ results
}}
```

```
10 equals 10 : true
20 equals 20 : false
```

Example 3 : toString()

```
public class ToString {
 public  static  void  main (String [] args)
{  Integer  num = 42;

  String str = num.toString();

 System.out.println ("String representation
                    of :" + num + " : " + str)

}}
```

```
String representation of 42 : 42
```