Objectives

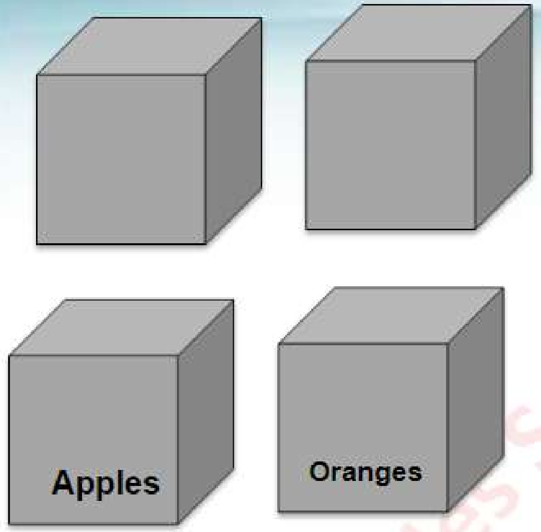
Click to Continue

This session will help you to understand the following,

- What are generics?
- Uses of generics.
- How to implement generics?
- How to iterate the collections.

Real World Analogy

Click to Continue



Assume that there are two boxes and one box contains Apples and other has oranges.

Solution: One can label the box. This way anyone who looks into the box will know easily know which box contains what fruit?

Similar to labelling a box, can we label the lists to let the compiler know what it stores?

Generics can be used to label the lists.

Generics

Click to Continue

- Generics allows programmer to specify the data type to be stored in a collection or a class.
- Compiler will throw an error if the data stored in collection is different from the data type specified in the generics.

Let us see how to declare collection with generics.

Syntax:

```
Collection Interface <Type> objectName=new
Implementation<Type>();
```

Illustration : Declares an array list for holding only String values.

```
List<String> myList=new ArrayList<String>();
```

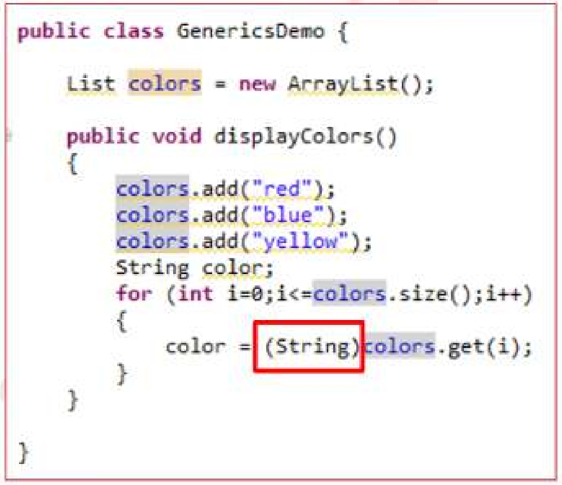
Advantages of generics in collections

Click to Continue

- When retrieving elements from collections typecast is not needed.

Illustration: ArrayList named colors, elements can be retrieved from the List as shown below

No Generics casting needed



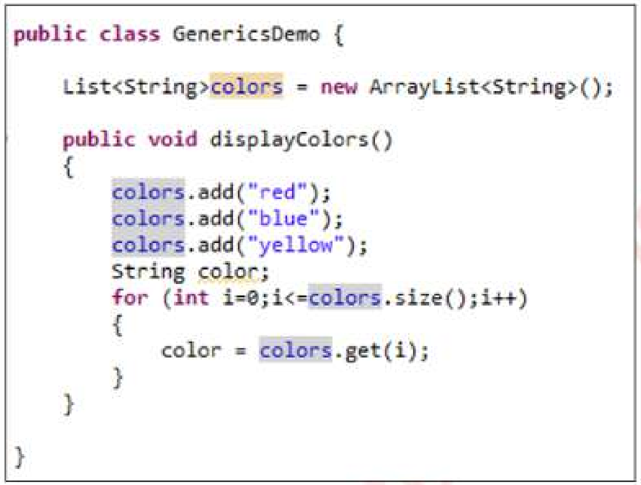
```
public class GenericsDemo {

    List colors = new ArrayList();

    public void displayColors()
    {
        colors.add("red");
        colors.add("blue");
        colors.add("yellow");
        String color;
        for (int i=0;i<=colors.size();i++)
        {
            color = (String)colors.get(i);
        }
    }

}
```

With Generics no casting needed



```
public class GenericsDemo {

    List<String>colors = new ArrayList<String>();

    public void displayColors()
    {
        colors.add("red");
        colors.add("blue");
        colors.add("yellow");
        String color;
        for (int i=0;i<=colors.size();i++)
        {
            color = colors.get(i);
        }
    }

}
```

Advantages of generics in collections

Click to Continue

- When retrieving elements from collections typecast is not needed.

  Illustration: ArrayList named colors, elements can be retrieved from the List as shown below

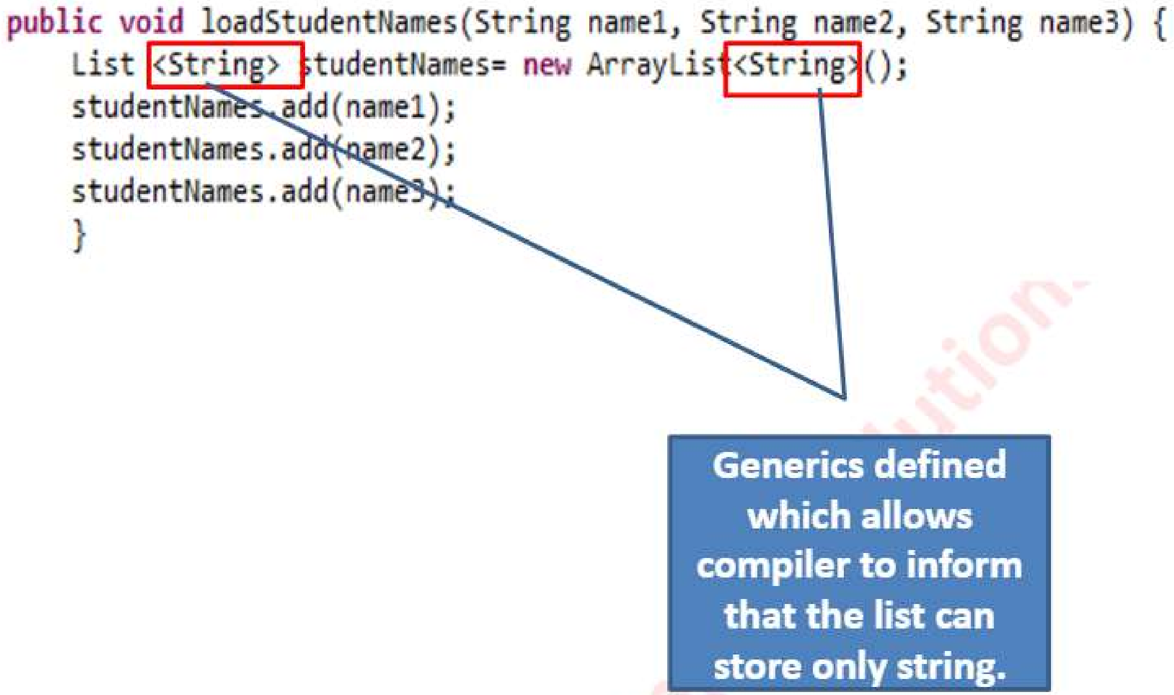
- No more class cast exceptions during run time when iterating through collections.
- Code will be **easily readable and maintainable:** Developers and data type stored in the collection can be easily understood by just looking at the code.

Try it out – Adding generics

Click to Continue

Let us take the same scenario we developed earlier and define generics.

Scenario 1: Develop a method **loadStudentNames** that accepts the names of three students as three string parameter and add them to an ArrayList. Define the list with generics String



```
public void loadStudentNames(String name1, String name2, String name3) {
    List <String> studentNames= new ArrayList<String>();
    studentNames.add(name1);
    studentNames.add(name2);
    studentNames.add(name3);
    }
```

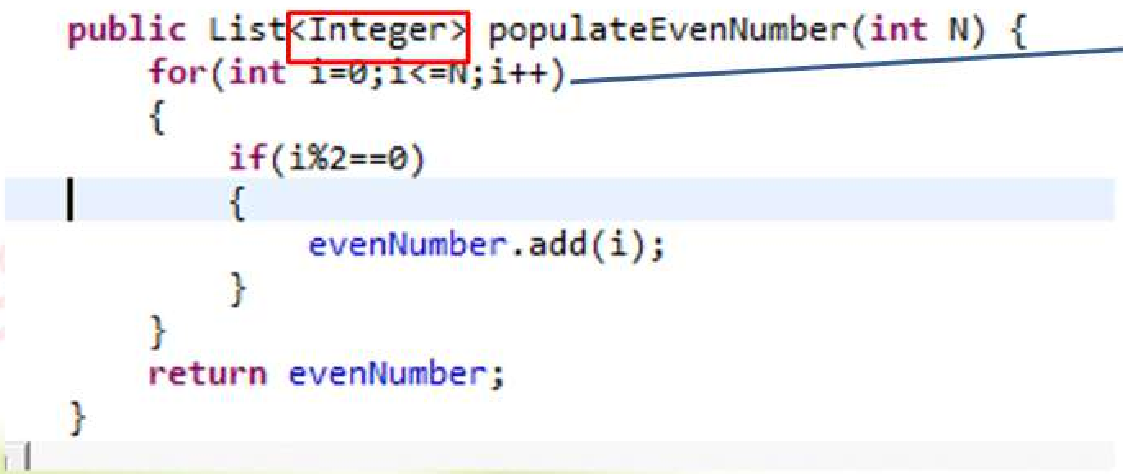
Generics defined which allows compiler to inform that the list can store only string.

Try loading Integer and check what happens?
studentNames.add(100);

Try it out – Adding Generics

Click to Continue

Scenario 2 : Create an method ***loadEvenNumbers*** which accepts a int N and iterates through 'N' even numbers add each number in the ArrayList and returns the list.. Define the list with generics Integer.



```
public List<Integer> populateEvenNumber(int N) {
    for(int i=0;i<=N;i++)
    {
        if(i%2==0)
        {
            evenNumber.add(i);
        }
    }
    return evenNumber;
}
```

Generics defined which allows compiler to inform that the list can store only Integer.

Iterating Collections

Click to Continue

Let us look at how to iterate collections

For loop

- Read collection element using **get() by passing index value.**
- Can be used only with List.

For-each loop

- Iterate over a list and fetches the elements. Index value not needed for retrieving the value
- Can be applied with both List and Set.

Iterator

- ***Iterator*** provides methods to iterate through a collection.
- Can be applied with both List and Set.

ListIterator

- An iterator which support both forward and back ward iteration.

Important: For each loops are commonly used in projects.

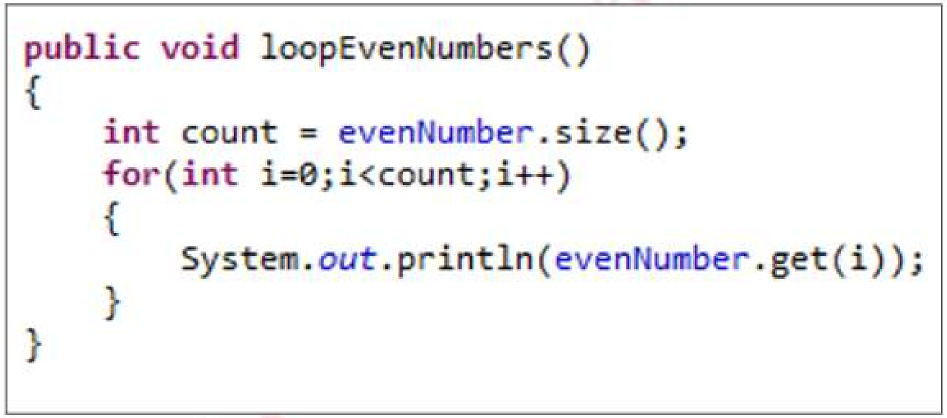
Try it out : For Loop

Click to Continue

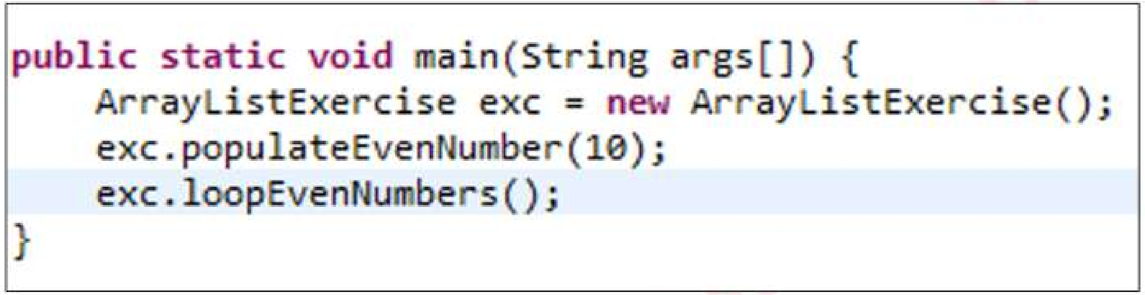
Let us now learn how too use for loop to iterate through a list. Let us reuse the same method we used in the previous example.

Scenario # 1 :

Write a method ***loopEvenNumber*** which iterates through the even number list and display. it.



```
public void loopEvenNumbers()
{
    int count = evenNumber.size();
    for(int i=0;i<count;i++)
    {
        System.out.println(evenNumber.get(i));
    }
}
```



```
public static void main(String args[]) {
    ArrayListExercise exc = new ArrayListExercise();
    exc.populateEvenNumber(10);
    exc.loopEvenNumbers();
}
```

This iterates through the even numbers using for loop and displays it.

Invoke the ***loopEvenNumber*** followed by **populateEvenNumber** method.

For-Each loop

Click to Continue

This is a feature added as part of Java 5, used to retrieve elements from a collection or array.

The advantage of *"for each"* is that the collection can be iterated without any index.

Syntax :

```
  for(datatype variableName : collectionName){
   loop body
   }
```

Illustration:

```
 for(Integer b : numberList){
 System.out.println(b);
 }
```

Reads the list *numberList* and retrieves the elements from collection as Integer.

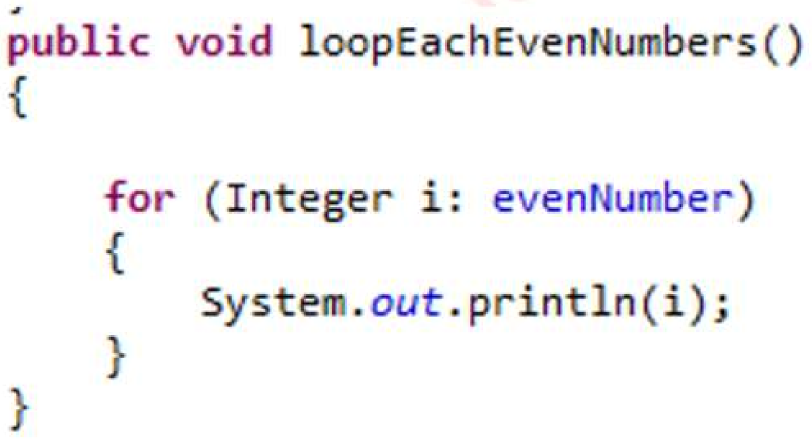
Try It Out – For Each loop

Click to Continue

Let us try for each loop to iterate the even numbers.

Scenario # 1 :

Write a method *loopEachEvenNumber* which iterates through the even number list and display it. This time use the for each loop.



```
public void loopEachEvenNumbers()
{

    for (Integer i: evenNumber)
    {
        System.out.println(i);
    }
}
```

Invoke the ***loopEach EvenNumber*** followed by **populateEvenNumber** method in the main method and see how looping works.

Iterator

Click to Continue

This interface used for iterating & accessing the elements of a collection. This can be used to iterate List and Set.

Iterator

Click to Continue

This interface used for iterating & accessing the elements of a collection. T e List and Set.

These are the iterator API's commonly used.

Method	Description
boolean hasNext()	true if there are more elements for the iterator.
Object next()	Returns the next object or elements.
void remove()	Removes the element that was returned by next from the collection. This method can be invoked only once per call to next .

Iterator

Click to Continue

This interface used for iterating & accessing the elements of a collection. Th List and Set.

Method	Description
boolean hasNext()	true if there are more elements for the iterator.
Object next()	Returns the next object or elements.
void remove()	Removes the element that was returned by next from the collection. This method can be invoked only once per call to next .

Syntax : Iterator<type> iteratorName=collection.iterator();

Where type is the generic type of the Iterator

Creates a iterator object from the collection

Illustration : Creates an iterator object for a given list numberList which has a list of numbers stored

```
Iterator<Integer> numberIterator = numberList.iterator();
```

Iterator implementation

Click to Continue

1 : Creates the Iterator object

```
Iterator<Integer> iterator=numberList.iterator();
```

2 : Use ***hasNext()*** method in while loop to check for element existence

```
while(iterator.hasNext()) {
 // Read Elements
}
```

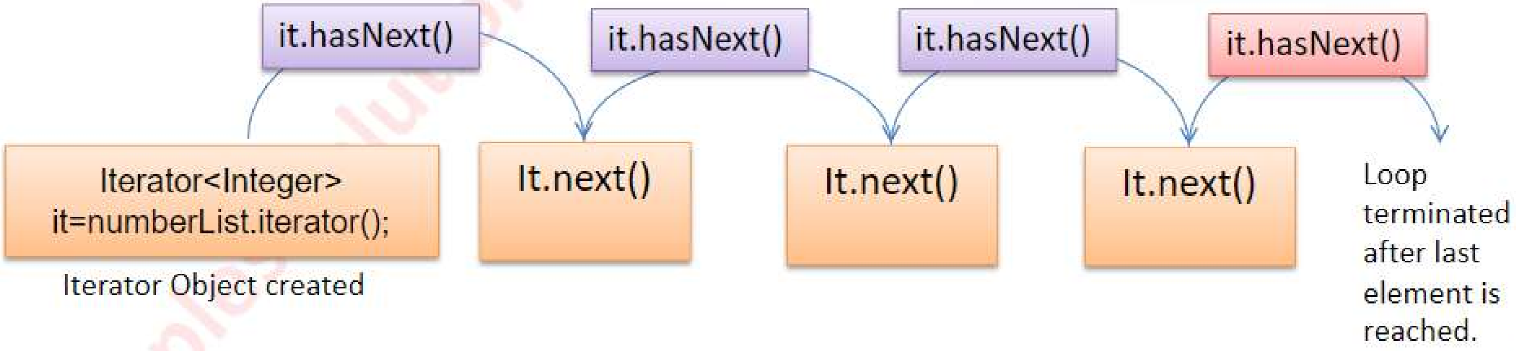
3 : Retrieve the element using the ***next()*** method.

```
while(iterator.hasNext()) {
 int number=iterator.next();
}
```

Iterator Behaviour

Click to Continue

Let us look at how iterator code works



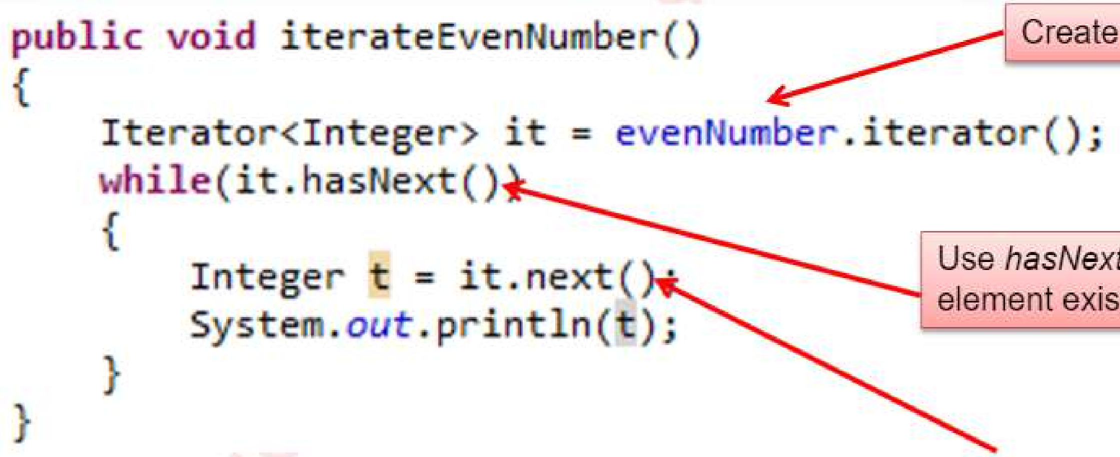
***hasNext()* -** is used to check the existence of next element.
***next()* -** element is used to fetch the next element in the list.

Try it out : Iterator

Click to Continue

We will use the exercise *loadEvenNumber to try iterator*

Write a method *iterateEvenNumber* which iterates through the even number list and display it. This time use iterator.



```
public void iterateEvenNumber()
{
    Iterator<Integer> it = evenNumber.iterator();
    while(it.hasNext())
    {
        Integer t = it.next();
        System.out.println(t);
    }
}
```

Create an iterator

Use *hasNext* method to check the element existence

Use the next() to fetch the next element in the collection.

Invoke the ***iterateEvenNumber*** followed by **populateEvenNumber** method in the main method and see how looping works.

You have completed the session