Session 5

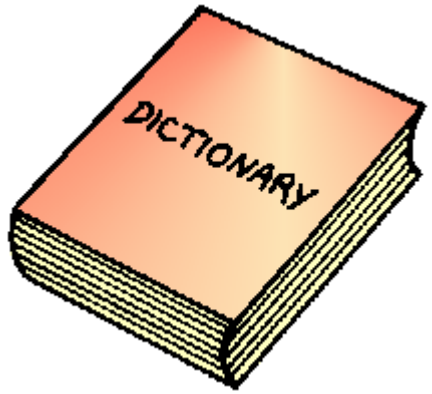
Collections and Exceptions in Dart

Session Overview

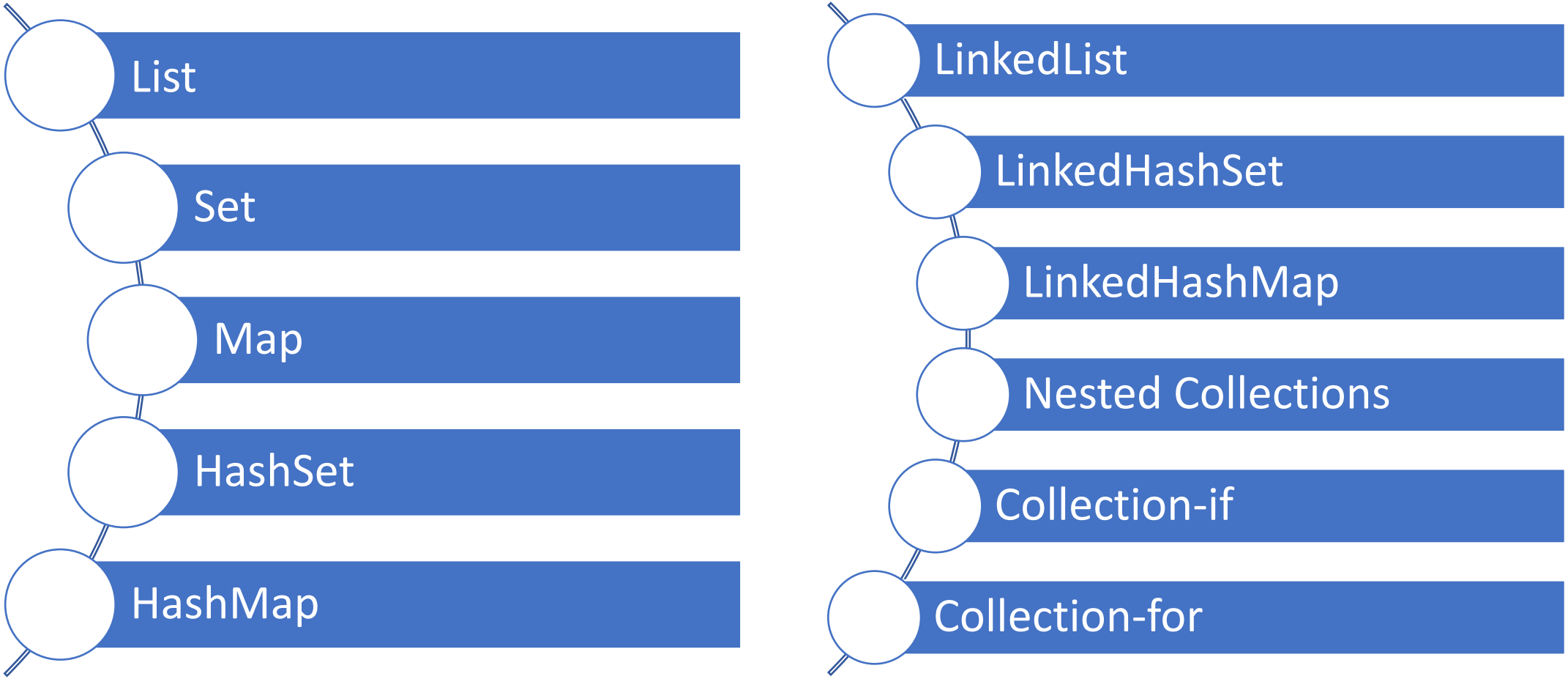
- Define a collection and its usage
- Explain types of collections in Dart
- Outline exceptions in Dart
- Explain exception handling in Dart
- Elaborate on errors in Dart

Collections

- Collections are used for storage, retrieval, and communication of mass data.
- Collections are a group of classes and interfaces that are highly optimized data structures.
- The `dart:core` library is imported using the `import` keyword to add collections to a Dart script.



Types of Collections



List

Fixed Length List

- Size is specified during initialization.
- Size cannot be modified later.

Growable Length List

- Specifying size is not mandatory during initialization.
- Size can be modified at runtime.

Code Snippet 1:

```
void main() {
   List list1 = List.filled(2, []);
   list1[0] = 50;
   list1[1] = 100;
   print(list1);
}
```

Code Snippet 2:

```
void main() {
    var newlist = [10,20,30,40,50];
    newlist.add(60);
    print(newlist);
}
```

Set and Map

A set is a unique, unordered collection of different values of the same data type.

Code Snippet 3:

```
void main() {
    Set s = new Set();
    s.add(10);
    s.add(20);
    s.add(30);
    s.add(40);
    s.add(40);
    print(s);
}
```

Code Snippet 4:

```
 var userdetails = {'email' : 'jack@gmail.com' ,
'Password' : '12345'};
 print(userdetails);
```

Code Snippet 5:

```
 var details = new Map();
 details['username'] = 'ron';
 details['password'] = '1234';
 print(details);
```

A map is a dynamic collection of all kinds of data types.

Data is stored in the form of key-value pairs, in which keys should be unique.

HashSet and HashMap [1-2]

HashSet is a hash table that has an unordered set implementation.

HashMap is a hash table map implementation that is unordered.

HashSet and HashMap are provided in the `dart:collection` library.

In both these collections, the order of retrieving may not be the same as the order in which they are inserted.

HashSet and HashMap [2-2]

Code Snippet 6: HashSet

```
import 'dart: collection';

void main() {
 Set numbers = new HashSet();
 numbers.add(10);
 numbers.add(20);
 numbers.add(30);
 numbers.add(40);
 numbers.add(50);
 print(numbers);
}
```

Code Snippet 7: HashMap

```
import 'dart: collection';


void main() {

 Map details = new HashMap();

 details['name'] = 'dan';
 details['email'] = 'dan@gmail.com';
 details['number'] = 'xxxxxxxxx';

 print(details);
}
```

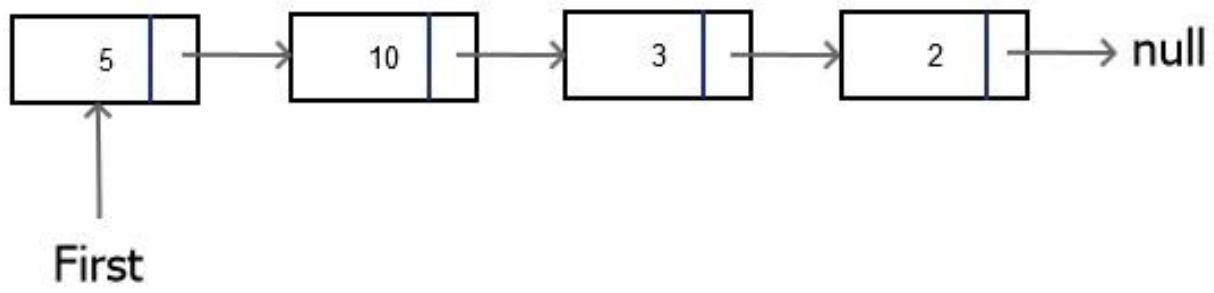
LinkedList

Code Snippet 8:

```
import 'dart:collection';
class Items extends LinkedListEntry<Items> {
    final int id;
    final String name;
    Items(this.id, this.name);
    @override
    String toString(){
        return '$id : $name';
    }
}

void main() {
     final linkedList = LinkedList<Items>();
     linkedList.addAll(
     [Items(1, 'jon'), Items(2, 'natalia'),
Items(3, 'dina')]);
     print(linkedList);
}
```

- A LinkedList is a sequential data structure.
- A LinkedList is unidirectional.
- A LinkedList supports faster insertions and removal of elements.



LinkedHashMap and LinkedHashSet

Both these data structures execute data in the same order as they are inserted.

While LinkedHashSet can store only one null value, LinkedHashMap can store one null key with multiple null values.

Code Snippet 9: LinkedHashMap

```
import 'dart:collection';

void main() {
 var l1 = new LinkedHashMap();
 l1['1'] = 'Alice';
 l1['2'] = 'Bob';
 l1['3'] = 'Cindy';
 l1['4'] = 'Alex';
 print(l1);
}
```

Code Snippet 10: LinkedHashSet

```
import 'dart:collection';
void main() {
 var l1 = new LinkedHashSet();
 l1.add('dan');
 l1.add('rony');
 l1.add('sam');
 print(l1);
}
```

Nested Collections

Collection of Collection

- It can be thought of as a multi-dimensional array.
- This is implemented using `List`, `Map`, `HashMap`, `HashSet`, and `LinkedList`.
- This concept is useful for creation of matrices.

Code Snippet 11:

```
import 'dart:collection';

void main() {

List l1 = List.generate(2, (_) => List.generate(2, (_) =>
0));

print(l1);

List l2 = List.generate(3, (_) => List.generate(3, (_) =>
0));

print(l2);

}
```

Collection-if and Collection-for

Collection-if

- Collection of objects can be checked for certain conditions using an `if` statement.

Collection-for

- Collection of objects can be iterated using `for` and `foreach` loops.

Code Snippet 12:

```
import 'dart:collection';
void main() {
    List l1 = new List();
    l1.add(10);
    l1.add(20);
    l1.add(30);
    l1.add(40);
    for(var v in l1){
     print(v);
    }
   if(l1[0] < l1[1]){
      l1[0] = l1[0] + 10;
      print(l1[0]);
   }
}
```

Exception Handling [1-2]

Every exception that occurs should be handled so that the program does not terminate abruptly.

All exceptions in Dart are a sub-type of the `Exception` class.

`try-catch/on` block

- The code that can cause exceptions is written in the `try` block.
- The `on` block is coded to handle a specific exception while the `catch` block is coded when a handler requires a generic `Exception` object.

`finally` block

- The code that has to be executed irrespective of exception occurrence is given in the `finally` block.
- This block is coded after the `try-catch/on` block.

`throw` clause

- The developer can raise an exception explicitly using the `throw` keyword.
- This exception should also be handled.

Exception Handling [2-2]

Code Snippet 13:

```
void main() {
 int a = 5;
 int b = 0;
 int res;

try {
 res = a~/b;
 }
 on IntegerDivisionByZeroException {
 print('Cannot divide by zero');
 } finally{
print('Finally block executed');
 }
}
```

Code Snippet 14:

```
void main() {
   void checkAge(int age) {
      if(age<0) {
         throw new FormatException();
      }
   }
   try {
     checkAge(-10);
    }
       catch(e) {
        print('Age cannot be negative');
    }
}
```

Built-in Exceptions

- DefferedLoadException
- FormatException
- IntegerDivisionByZeroException
- IOException
- Timeout

Errors

- Errors occur when there is an unexpected problem that can be fixed by the programmer.
- Some examples are syntax errors, null check, and use of invalid methods.

Errors	Exceptions
Errors may occur due to lack of system resources.	Exceptions occur due to application code.
Errors that occur at runtime are not known by the compiler.	Exceptions may or may not be caught by the compiler.
Errors are unchecked.	Exceptions can be checked or unchecked.

Summary

- A collection is an entity that represents a group of elements that can be of any data type.
- The `dart:core` library enables support for collections in Dart and can be appended to the Dart script using the `import` keyword.
- Collections are utilized to store, retrieve, and communicate mass data. They represent data items that form a natural cluster such as a mail folder (assortment of letters) or a directory (a mapping of names to phone numbers).
- An exception is an abnormal condition that disrupts the natural rhythm of execution in a program. It may occur during execution time (runtime).
- When an exception is raised, it stops other parts of the program as well. Hence, Dart has a methodology to handle such conditions by utilizing exception handlers. This is called as exception handling.