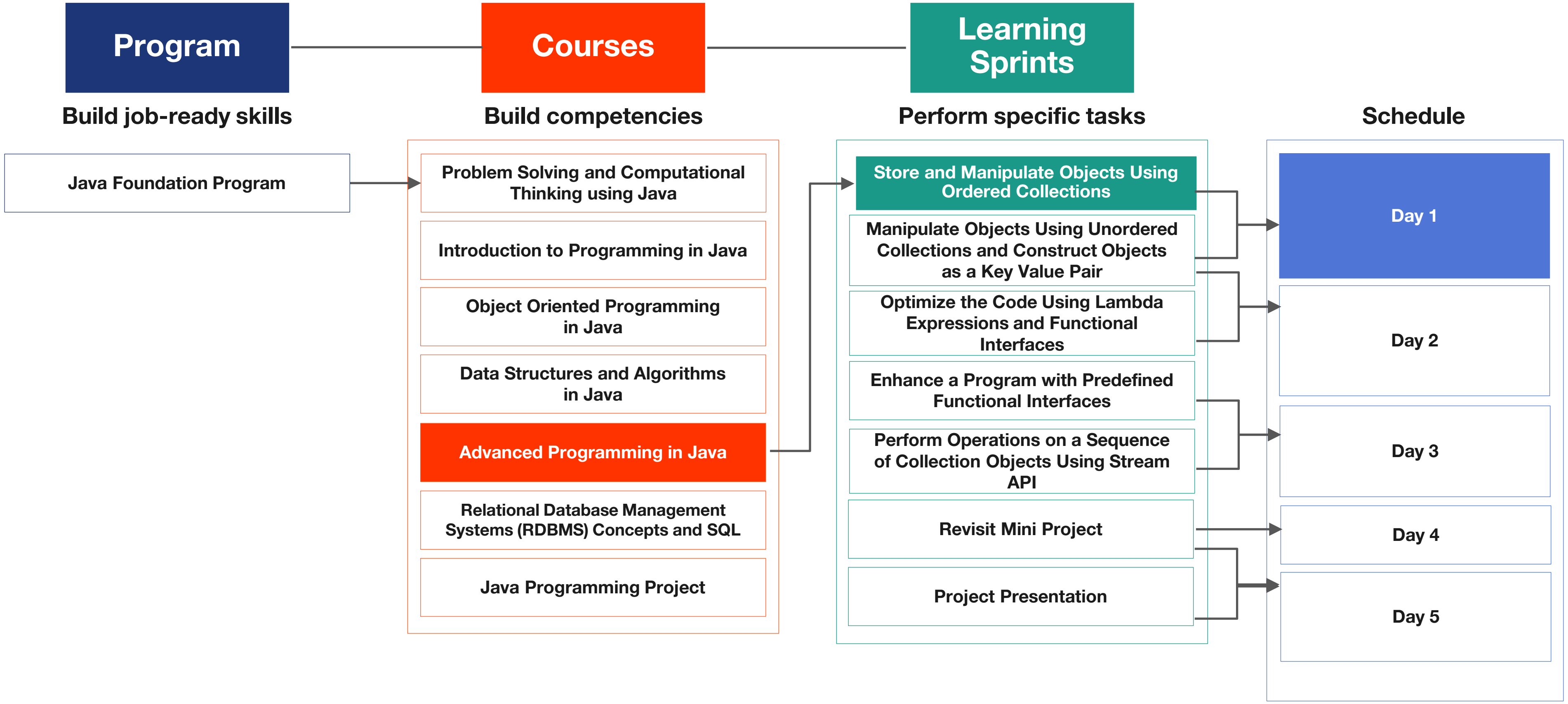
Java Program: Course 5: Plan



Program — Build job-ready skills

- Java Foundation Program

Courses — Build competencies

- Problem Solving and Computational Thinking using Java
- Introduction to Programming in Java
- Object Oriented Programming in Java
- Data Structures and Algorithms in Java
- **Advanced Programming in Java**
- Relational Database Management Systems (RDBMS) Concepts and SQL
- Java Programming Project

Learning Sprints — Perform specific tasks

- **Store and Manipulate Objects Using Ordered Collections**
- Manipulate Objects Using Unordered Collections and Construct Objects as a Key Value Pair
- Optimize the Code Using Lambda Expressions and Functional Interfaces
- Enhance a Program with Predefined Functional Interfaces
- Perform Operations on a Sequence of Collection Objects Using Stream API
- Revisit Mini Project
- Project Presentation

Schedule

- **Day 1**
- Day 2
- Day 3
- Day 4
- Day 5

Think and Tell

A sports academy wishes to analyze the performance of the top 10 players on the basis of their age.

How do you think a programmer will represent the data containing the age details of the top 10 players?

Ages of Players

Declare an array with 10 items indexed from 0 to 9:

```
int ageOfPlayer[ ] = new int[10]
```

Player Details

Now the sports academy wishes to represent the name, age, height, weight, and country details of the players within their application.

How can we achieve this within our application?

Can we represent all these details within a single array of primitive type?

Player Details

As the number of new players is growing, the sports academy wishes to add the details of the new players, as and when they join the academy.

What changes are required to do this within the current code of the application?

Player Details

It is possible that over a period of time, the players might leave the academy.

If someone leaves the academy then the details of that player have to be removed.

Can we keep adding and removing details as and when required?

Player Details

The sports academy also wishes to represent these details according to the names in an ascending order.

Is arranging data possible in an Array structure?

If yes, then will it be efficient and flexible?

Player Details

Can we use Linked List to achieve this objective? If yes, then what are the changes we need to make within the existing code?

Will this code be implemented using the defined standards and best practices?

Will it be easy to debug?

Let Us Discuss

Consider one more scenario where we need to develop a class with a method, which can return various objects, such as an Integer, a String, and a Double.

How do we achieve this objective within our Java program? Can we create a class and method inside the class which can use any type of data, and return any type of data as per the requirement?

How can we ensure type safety with the class created by us?

Store and Manipulate Objects Using Ordered Collections

Learning Objectives

- Define and create generic classes
- Implement generic classes and methods
- Explain Java collections and its hierarchy
- Describe and implement different types of list interfaces

Generic Classes

- Generics enable you to generalize classes
- In the declaration of a generic class, the name of the class is followed by a type parameter section
- A generic class:
  - Provides type safety to code
  - Moves many errors from runtime to compile time
  - Provides cleaner and easier-to-write codes
  - Reduces the need for casting with collections

Create Generic Classes

- The following syntax is used to create generic classes:

```
class [ClassName]<T>
{
}
```

- You can create a generic class by using the following code snippet:

```
class GenericClassDemo<T>
{
}
```

- The type parameter section is represented by angular brackets,< >, that can have one or more types of parameters separated by commas

Create Generic Classes (contd.)

- In case you want to create a generic class to set and get the Integer and String values, you can use the following code:

```
public class GenericClassDemo<T>
{
        private T t;

        public void setValue(T t)
        {
                this.t = t;
        }
        public T getValue()
        {
                return t;
        }
}
```

Here, a variable named T has been added to the class definition surrounded by the angular <> brackets. This T variable stands for “type” and can represent any type.

Create Generic Classes (contd.)

```
public static void main(String[] args)
     {
        GenericClassDemo<Integer>  iobj = new GenericClassDemo<Integer>();
         iobj.setValue(10);
         System.out.println(iobj.getValue());

         GenericClassDemo<String> sobj = new GenericClassDemo<String>();
         sobj.setValue("Ten");
         System.out.println(sobj.getValue());
     }
}
```

Now, the code can be changed to use T type instead of specific type information. So, the GenericClassDemo class can store any type of object.

Quick Check!

What are the advantages of using generic classes?

Create a Generic Method

- In a generic class, a method can use the type parameter of the class, which automatically makes the method generic

- Consider the following code of the generic method inside the generic class:

```
 public class GenericClassDemo<T>
{
        private T t;
        public void setValue(T t)
        {
                        this.t = t;
        }
        public T getValue()
        {
                        return t;
        }
}
```

Create a Generic Method (contd.)

- The declaration of a generic method contains the type parameter that is represented by angular brackets, < >.
- The following syntax is used to create a generic method:

```
public <Type Parameter> [Return Type] [MethodName](Argument list…)
{

}
```

- You can create a generic method, as shown in the following code snippet:

```
public <T> T showValue(T val)
{

}
```

Create a Generic Method (contd.)

- Consider the following code of the generic method inside the non-generic class:

```
public class GenericMethodDemo{
    public <M> M display(M val){
        return val;
    }
    public static void main(String[] args){
        GenericMethodDemo obj = new GenericMethodDemo();
System.out.println("The generic method is called with String value: "+
obj.display("Test"));
System.out.println("The generic method is called with Double value: " +
obj.display(7.5));
System.out.println("The generic method is called with Boolean value: " +
obj.display(true));
System.out.println("The generic method is called with Integer value: " +
obj.display(10));
    }}
```

Quick Check!

Which one of these type parameters is used for a generic method to return and accept any type of object?

1. K
2. N
3. [✓ T]
4. V

Quick Check!

Identify the correct syntax of the generic method.

1. <T1, T2, …, Tn> name(T1, T2, …, Tn) { /* … */ }

2. public <T1, T2, …, Tn> name<T1, T2, …, Tn> { /* … */ }

3. class <T1, T2, …, Tn> name[T1, T2, …, Tn] { /* … */ }

4. <T1, T2, …, Tn> name{T1, T2, …, Tn} { /* … */ }

Implement Type-Safety

- To create an object of a generic class, you need to invoke the constructor of a generic class with the required type of parameters
- Java provides the flexibility to leave the type parameters empty as long as the compiler can guess or judge the type of the argument from the context
- Java provides wildcards that allow you to achieve inheritance in a type parameter

Using a Generic Type Inference

- While using a generic type, you must provide the type argument for every type parameter declared for the generic type
- The type argument list is a comma-separated list that is delimited by angular brackets and follows the type name
- Java provides a new feature called the type inference
- Type inference enables you to invoke the constructor of a generic class with an empty set of type parameters, <>
- The empty set of type parameters can be used as long as the compiler can infer the type arguments from the context

Using Wildcards

- In generics, a subclass cannot be passed as a subtype of a superclass, as shown in the following code snippet:

```
WildCardDemo<Number> obj = new WildCardDemo<Integer>();
```

- Generics only allow the following type of declaration:

```
WildCardDemo<Integer> obj = new WildCardDemo<Integer>();
Or,
WildCardDemo<Number> obj = new WildCardDemo<Number>();
```

- Some of the commonly used wildcard arguments are:
  - ? extends
  - ? super
  - ?

Quick Check!

Which wildcard is used to specify the type inherited from the specified class?

1. <?>
2. <? extends >
3. <extends ?>
4. <? extends ?>

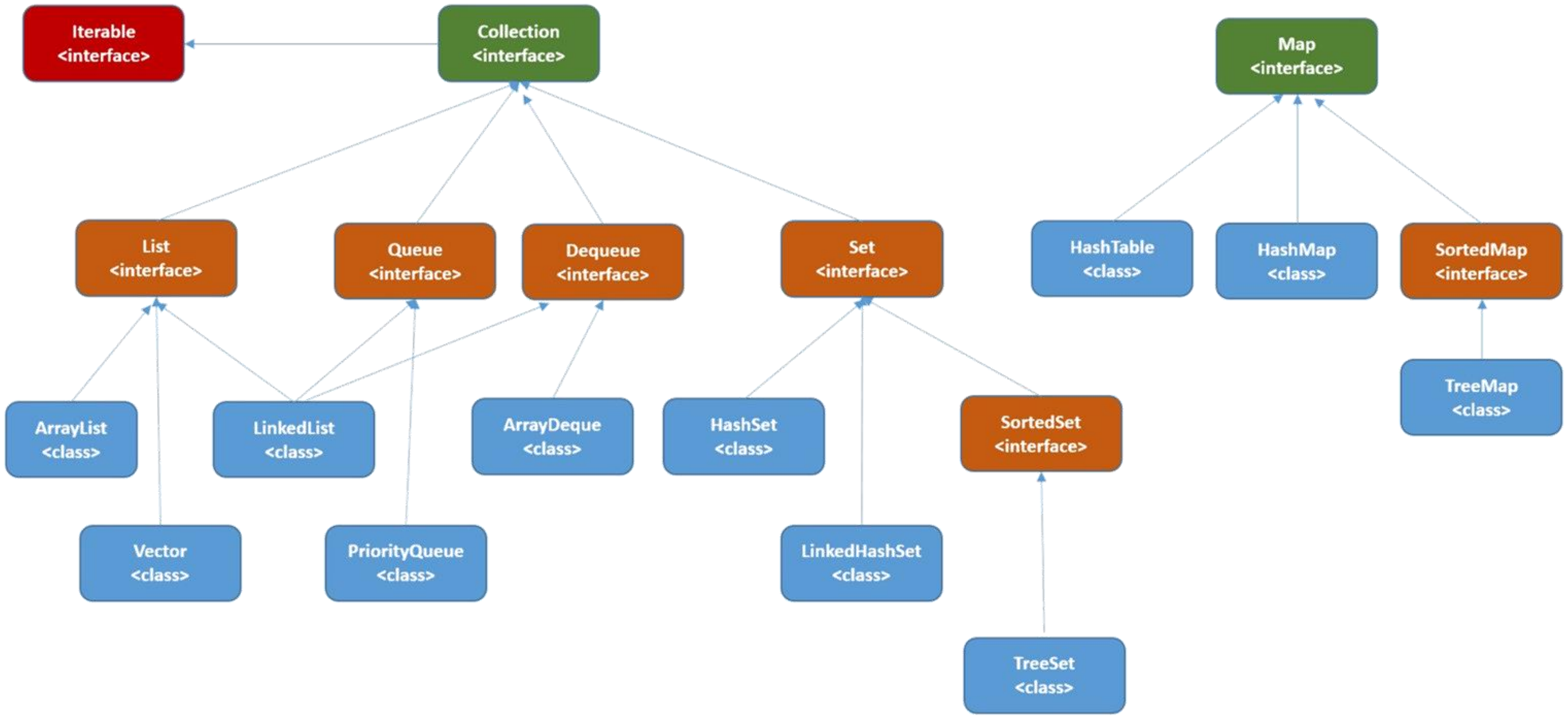
Numeric Objects and Wildcard Characters

Write a program in Java to compare only the numeric objects using generic class and wildcard characters.

Java Collections

- A collection is an object that contains a group of objects within it; these objects are called the elements of a collection that descend from a common parent type
- Collections have an advantage over arrays; they can grow to any size unlike arrays
- They do not hold primitive types
- They implement many data structures including stack, queue, dynamic array, and hash
- Collections API:
  - Relies on generics for implementation
  - Classes are stored in the java.util package

Java Collections Framework Hierarchy



Using the List Interface

- The List interface is used to create an ordered collection of objects, which can contain duplicate objects
- In this collection, the objects can be inserted one after the other or at the specified position in the list
- The package, java.util, provides the ListIterator interface to traverse through the list collection
- The reference of the ListIterator interface can be obtained using the listIterator() method of the List interface
- The ListIterator interface contains various methods, such as hasNext(), hasPrevious(), next(), and previous(), which help to traverse the list in both directions

Working with the ArrayList Class

- The ArrayList class provides the implementation of the List interface. It enables you to create a resizable array. It is efficient in searching an object in the list and inserting the objects at the end of the list

- Some of the commonly used constructors of the ArrayList class are:
  - ArrayList()
  - ArrayList(Collection<? extends E> c)

- Some of the commonly used methods of the ArrayList class are:
  - boolean add(E e)
  - void clear()
  - E get(int index)
  - E remove(int index)
  - boolean remove(Object o)
  - int size()

Interactive Demo

Write a program in Java to add, traverse and remove objects from an ArrayList.

Working with the LinkedList Class

- The LinkedList class provides the implementation of the List interface. It enables you to create a doubly-linked list. It allows you to traverse in the forward or backward direction

- Some of the commonly used constructors of the LinkedList class are:
  - LinkedList()
  - LinkedList(Collection<? extends E> c)

- Some of the commonly used methods of the LinkedList class are:
  - boolean add(E e)
  - void clear()
  - E get(int index)
  - E remove(int index)
  - boolean remove(Object o)
  - int size()

Interactive Demo

Write a program in Java to add, traverse and remove objects from a LinkedList.

Working with the Vector Class

- The Vector class is similar to the ArrayList and LinkedList classes. The methods of the Vector class are synchronized.
- Some of the commonly used constructors of the Vector class are:
  - Vector()
  - Vector(Collection<? extends E> c>
  - Vector(int initialCapacity)
- Some of the commonly used methods of the Vector class are:
  - boolean add(E e)
  - void clear()
  - E get(int index)
  - E remove(int index)
  - boolean remove(Object o)
  - int size()

Interactive Demo

Write a program in Java to add, traverse and remove objects from Vector.

Quick Check!

Which one of the following statements holds true about Collection classes?

1. ArrayList implements a Set collection.
2. List is a collection that can be used to implement a stack or a queue.
3. The Collection classes are all stored in the java.lang package.
4. ArrayList is a dynamically growable array.

Key Takeaways

- Explain generics
- Implement a generic class
- Illustrate Java collections
- Determine the List interface

Thank you!