# Experiment:6

## Generics:

1. **Declare a class InvoiceDetail which accepts a type parameter which is of type Number with following data members class InvoiceDetail <N extends Number> {**
   **private String invoiceName;**
   **private N amount;**
   **private N Discount;**
   **// write getters, setters and constructors Call the methods in Main class**
   **}**

Solution:

```
class InvoiceDetail<N extends Number> {
   private String invoiceName;
   private N amount;
   private N discount;

   // Constructor
   public InvoiceDetail(String invoiceName, N amount, N discount) {
      this.invoiceName = invoiceName;
      this.amount = amount;
      this.discount = discount;
   }

   // Getters and Setters
   public String getInvoiceName() {
      return invoiceName;
   }

   public void setInvoiceName(String invoiceName) {
      this.invoiceName = invoiceName;
   }
   public N getAmount() {
      return amount;
   }
   public void setAmount(N amount) {
      this.amount = amount;
```

```
    }
    public N getDiscount() {
        return discount;
    }
    public void setDiscount(N discount) {
        this.discount = discount;
    }

    // Display Invoice Details
    public void display() {
        System.out.println("Invoice Name: " + invoiceName);
        System.out.println("Amount: " + amount);
        System.out.println("Discount: " + discount);
    }
}

// Main class to test InvoiceDetail
public class Main {
    public static void main(String[] args) {
        InvoiceDetail<Integer> invoice1 = new InvoiceDetail<>("Laptop Purchase", 50000,
5000);
        InvoiceDetail<Double> invoice2 = new InvoiceDetail<>("Smartphone Purchase",
39999.99, 3999.99);
        invoice1.display();
        System.out.println();
        invoice2.display();
    }
}
```

```
PS C:\12302130501036> javac rem.java
PS C:\12302130501036> java rem
Invoice Name: Laptop Purchase
Amount: 50000
Discount: 5000

Invoice Name: Smartphone Purchase
Amount: 39999.99
Discount: 3999.99
```

## 2.Implement Generic Stack

Solution:

```
import java.util.ArrayList;
```

```
class GenericStack<T> {
   private ArrayList<T> stack;
   public GenericStack() {
      stack = new ArrayList<>();
   }

   // Push element onto stack
   public void push(T item) {
      stack.add(item);
   }

   // Pop element from stack
   public T pop() {
      if (isEmpty()) {
         throw new RuntimeException("Stack is empty!");
      }
      return stack.remove(stack.size() - 1);
   }

   // Peek at top element
   public T peek() {
      if (isEmpty()) {
         throw new RuntimeException("Stack is empty!");
      }
      return stack.get(stack.size() - 1);
   }

   // Check if stack is empty
   public boolean isEmpty() {
      return stack.isEmpty();
   }
}

// Main class to test GenericStack
public class Main {
   public static void main(String[] args) {
```

```
        GenericStack<Integer> intStack = new GenericStack<>();
        intStack.push(10);
        intStack.push(20);
        intStack.push(30);
        System.out.println("Top Element: " + intStack.peek());
        System.out.println("Popped Element: " + intStack.pop());
        System.out.println("Is Stack Empty? " + intStack.isEmpty());
    }
}
```

Output:

```
PS C:\12302130501036> javac rem.java
PS C:\12302130501036> java rem
Top Element: 30
Popped Element: 30
Is Stack Empty? false
```

## 3.Write a program to sort the object of Book class using comparable and comparator interface. (Book class consist of book id, title, author and publisher as data members)

Solution:

```
import java.util.*;

// Book class implementing Comparable
class Book implements Comparable<Book> {
    private int bookId;
    private String title;
    private String author;
    private String publisher;

    // Constructor
    public Book(int bookId, String title, String author, String publisher) {
        this.bookId = bookId;
        this.title = title;
        this.author = author;
        this.publisher = publisher;
    }

    // Getters
```

```
    public int getBookId() {
        return bookId;
    }
    public String getTitle() {
        return title;
    }
    public String getAuthor() {
        return author;
    }
    public String getPublisher() {
        return publisher;
    }

    // Sorting by Title (Natural Order)
    @Override
    public int compareTo(Book other) {
        return this.title.compareTo(other.title);
    }

    // Display Book Details
    @Override
    public String toString() {
        return "Book ID: " + bookId + ", Title: " + title + ", Author: " + author + ", Publisher: "
+ publisher;
    }
}

// Comparator for sorting by Author
class AuthorComparator implements Comparator<Book> {
    @Override
    public int compare(Book b1, Book b2) {
        return b1.getAuthor().compareTo(b2.getAuthor());
    }
}

// Main class to test sorting
public class Main {
```

```
    public static void main(String[] args) {
        List<Book> books = new ArrayList<>();
        books.add(new Book(101, "Java Programming", "James Gosling", "TechPress"));
        books.add(new Book(102, "Data Structures", "Robert Lafore", "Pearson"));
        books.add(new Book(103, "Operating Systems", "Andrew Tanenbaum", "Prentice
Hall"));
        books.add(new Book(104, "Machine Learning", "Tom Mitchell", "McGraw Hill"));

        // Sorting by Title (Comparable)
        Collections.sort(books);
        System.out.println("Books sorted by Title:");
        for (Book book : books) {
            System.out.println(book);
        }

        // Sorting by Author (Comparator)
        Collections.sort(books, new AuthorComparator());
        System.out.println("\nBooks sorted by Author:");
        for (Book book : books) {
            System.out.println(book);
        }
    }
}
```

Output:

```
PS C:\12302130501036> javac rem.java
PS C:\12302130501036> java rem
Books sorted by Title:
Book ID: 102, Title: Data Structures, Author: Robert Lafore, Publisher: Pearson
Book ID: 101, Title: Java Programming, Author: James Gosling, Publisher: TechPress
Book ID: 104, Title: Machine Learning, Author: Tom Mitchell, Publisher: McGraw Hill
Book ID: 103, Title: Operating Systems, Author: Andrew Tanenbaum, Publisher: Prentice Hall

Books sorted by Author:
Book ID: 103, Title: Operating Systems, Author: Andrew Tanenbaum, Publisher: Prentice Hall
Book ID: 101, Title: Java Programming, Author: James Gosling, Publisher: TechPress
Book ID: 102, Title: Data Structures, Author: Robert Lafore, Publisher: Pearson
Book ID: 104, Title: Machine Learning, Author: Tom Mitchell, Publisher: McGraw Hill
```