

## Multiple Choice Questions (MCQs)

### Java Arrays :

**Q1:** Which of the following initializes an integer array with 5 elements?

ANS: A

**Q2:**What is the default value of an integer array element in Java?

ANS: B

**Q3:**How can we determine the size of an array arr in Java?

ANS: B

**Q4:**Which of the following correctly declares and initializes a 2D array?

ANS: A

**Q5:**Arrays in Java are:

ANS: D

### Java OOP :

**Q6:**Which of the following principles best describes encapsulation?

ANS: A

**Q7:**In Java, multiple inheritance is achieved through:

ANS: B

**Q8:**What is the output of the following code?

ANS: B

**Q9:**What is polymorphism in Java?

ANS: D

**Q10:**Which of the following is true about the this keyword?

ANS: A

### **Java Control Structures :**

**Q11:**Which of these control structures is not available in Java?

ANS: C

**Q12:**How many times will the following loop execute?

ANS: B

**Q13:**Which keyword is used to exit a loop early?

ANS: C

**Q14:**What will be the output of the following code?

ANS: A

### **Git :**

**Q15:**What does the command git clone do?

ANS: C

**Q16:**How do you undo the last commit without removing the changes in your working directory?

ANS: B

**Q17:**Which command displays the commit history?

ANS: B

**Q18:**Which command stages files for commit?

ANS: A

### **JDBC:**

**Q19:**Which JDBC driver type is known as the "pure Java" driver?

ANS: D

**Q20:**What method is used to execute an SQL query that returns data?

ANS: B

**Q21:**Which method is used to close a JDBC connection?

ANS: A

## Case Study 1: Banking System (OOP)

**Question:** Design a banking application where users can create accounts, check their balance, deposit, and withdraw funds. Use OOP principles to structure the solution, describing how classes like Account, Transaction, and Bank might interact.

**ANS:**

```
package assignments;

import java.util.HashMap;
import java.util.Map;

class account{
    private int no;
    private String name;
    private int balance;
    public account(int no,String name,int balance) {
        this.no=no;
        this.name=name;
        this.balance=balance;
    }
    public int getNo() {
        return no;
    }

    public String getName() {
        return name;
    }

    public int getBalance() {
        return balance;
    }

    public void deposit(int amount) {
        if(amount<=0) {
            System.out.println("enter valid amount");
        }
        else {
            balance =balance+amount;
            System.out.println("amount deposited : "+amount);

            System.out.println("current balance : "+balance);
        }
    }
    public void withdraw(int amount) {
        if(amount<=0) {
            System.out.println("enter a valid amount");
        }
        else if(amount>balance) {
            System.out.println("insufficient balance");
        }
        else {
            balance =balance-amount;
            System.out.println("amount withdraw : "+amount);
            System.out.println("current balance : " +balance);
        }
    }
    public String toString() {
```

```

        public String toString() {
            return "Account{" +
                "accountnumber='" + no + '\'' +
                ", name='" + name + '\'' +
                ", balance=" + balance +
                '}';
        }
    }

    class transaction{
        String transactiontype;
        int amount;
        account a;
        public transaction (String transactiontype,int amount,account a) {
            this.transactiontype=transactiontype;
            this.amount=amount;
            this.a=a;
        }
        public void trans() {
            if(transactiontype.equals("deposit")) {
                a.deposit(amount);
            }
            else if(transactiontype.equals("withdraw")) {
                a.withdraw(amount);
            }
        }
    }

}

}

class bank{
    private Map<Integer, account> accounts= new HashMap<>();

    public void createAccount(int no, String name, int initialBalance) {

        account a = new account(no, name, initialBalance);
        accounts.put(no, a);
        System.out.println("Account created: " + a);
    }

    public account getAccount(int no) {
        return accounts.get(no);
    }

    public void executeTransaction(int no, String transactiontype, int amount) {
        account a = getAccount(no);

        transaction t = new transaction(transactiontype, amount, a);
        t.trans();
    }
}

```

```

        if (a != null) {
            System.out.println(a);
        } else {
            System.out.println("Account not found.");
        }
    }
}

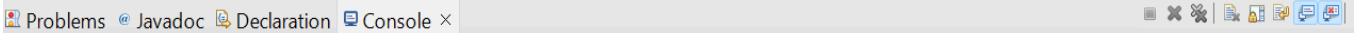
}

public class bankingsystem {

    public static void main(String[] args) {
        account a = new account(10,"rajitha",10000);
        transaction t = new transaction("deposit",1000,a);
        t.trans();
        transaction t1 = new transaction("withdraw",2000,a);
        t1.trans();
        bank b =new bank();
        b.createAccount(20,"rajitha", 1000);
        System.out.println( b.getAccount(20));
        b.executeTransaction(20,"deposit", 100);

    }
}

```



Problems @ Javadoc Declaration Console x  
 <terminated> bankingsystem (2) [Java Application] C:\Users\Admin\Downloads\eclipse-java-2024-06-R-win32-x86\_64\eclipse\plugins\org.eclipse.justj.openjdk.h  
 amount deposited : 1000  
 current balance : 11000  
 amount withdraw : 2000  
 current balance : 9000  
 Account created: Account{accountnumber='20', name='rajitha', balance=1000}  
 Account{accountnumber='20', name='rajitha', balance=1000}  
 amount deposited : 100  
 current balance : 1100

## Process:

- Created account, transaction and bank classes and added respective attributes to each class
- Written deposit and withdraw methods in account class which contains the logic to add and withdraw amount
- Since accounts has to be stored i used collections and created Maps and created methods to add accounts created to this map

- Later added transaction method to bank class so that everything can be executed at one place

## Output:

As seen in above output screenshot amount deposited and balance after the amount disposition is done are displayed also after account is created it is stored in map and the added account details are displayed

## Case Study 2: Git in Team Collaboration

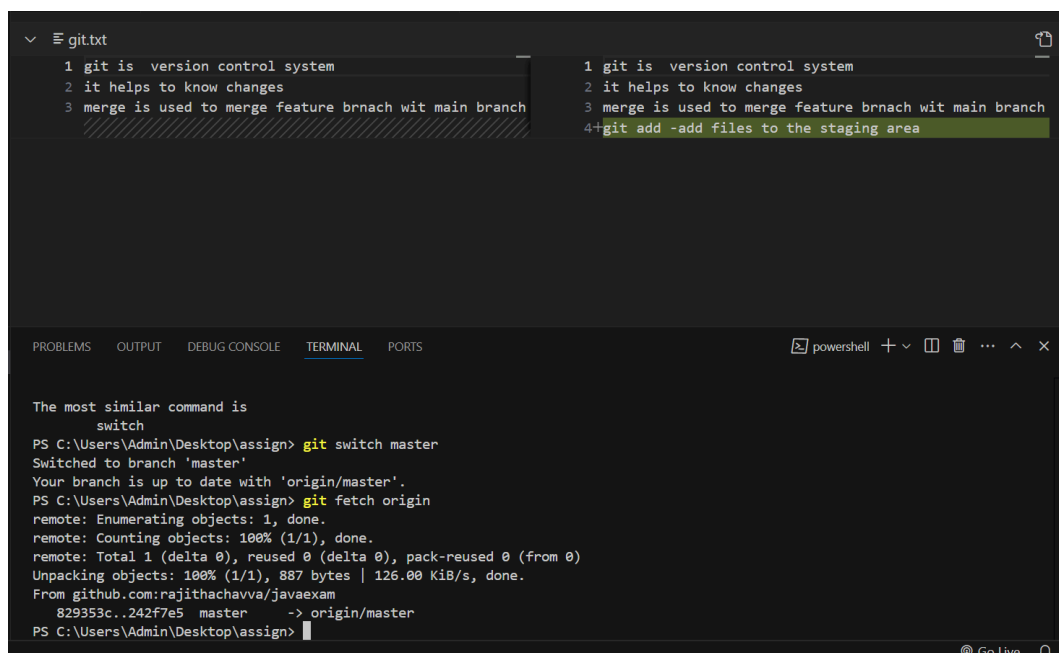
**Question: A team of 5 developers is working on a feature branch. Describe the process they should follow to keep their changes synchronized, using git fetch, git pull, and git merge. Discuss potential merge conflicts and how they could be resolved.**

### ANS:

To Ensure that each developer is working on the latest version of the code before they start their work we can use several git commands to keep the work synchronised with latest code

There are several ways to achieve it such as git fetch, git pull and git merge

**1. git fetch** command updates your local references to the remote repository, but it does not merge the changes into your working branch. This gives developers a chance to review changes before integrating them.



```

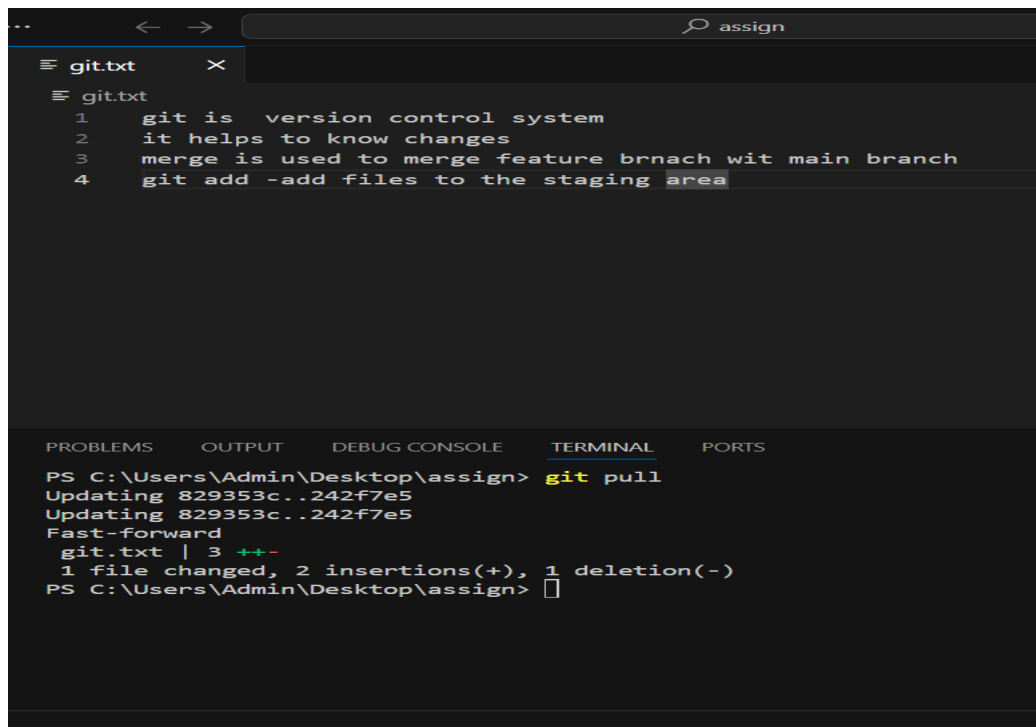
1 git is version control system
2 it helps to know changes
3 merge is used to merge feature brnach wit main branch
4+git add -add files to the staging area

The most similar command is
switch
PS C:\Users\Admin\Desktop\assign> git switch master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
PS C:\Users\Admin\Desktop\assign> git fetch origin
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (1/1), 887 bytes | 126.00 KiB/s, done.
From github.com:rajithachavva/javaexam
 829353c..242f7e5 master    -> origin/master
PS C:\Users\Admin\Desktop\assign>

```

As displayed in above picture git fetch just displays the changes and will not merge them this way developer can review the changes before merging them

2. **Git pull** is a combination of fetch and merge; it not only downloads the changes but also merges them .

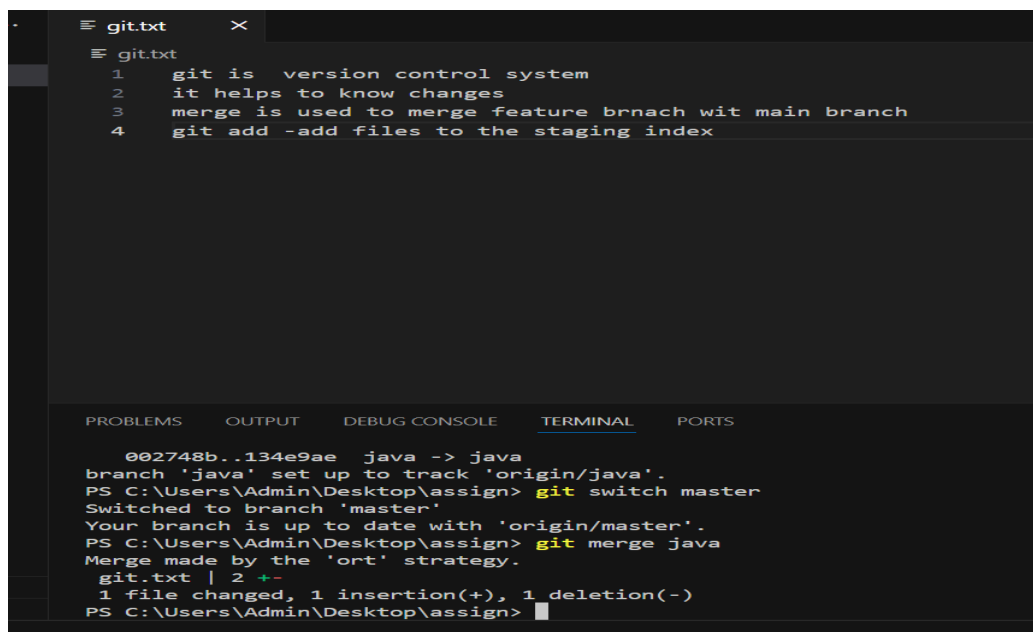


```
git.txt
1  git is version control system
2  it helps to know changes
3  merge is used to merge feature brnach wit main branch
4  git add -add files to the staging area

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Admin\Desktop\assign> git pull
Updating 829353c..242f7e5
Updating 829353c..242f7e5
Fast-forward
 git.txt | 3 ++-
1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\Admin\Desktop\assign>
```

Here in this picture we can see that git pull merges the changes to the current branch unlike fetch which just displays the changes this way before working on a code developer can ensure the code is in the latest version .

3.git merge command is used to merge feature branch back into main branch



```
git.txt
1  git is version control system
2  it helps to know changes
3  merge is used to merge feature brnach wit main branch
4  git add -add files to the staging index

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
002748b..134e9ae java -> java
branch 'java' set up to track 'origin/java'.
PS C:\Users\Admin\Desktop\assign> git switch master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
PS C:\Users\Admin\Desktop\assign> git merge java
Merge made by the 'ort' strategy.
 git.txt | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\Admin\Desktop\assign>
```



For example if a developer used a specific branch to push the changes to github and then created a pull request to merge into master , in this situation we can either do git pull to update our local repository into latest version or we can simply merge the branch( that developer used to push the changes into master

We can see in above picture that java branch is created to push changes so instead of doing git pull we directly used git merge java to merge the changes into master

### **Potential Merge Conflicts and How to Resolve Them:**

Merge conflicts usually happen when two developers have edited the same lines of code in the same file , when there are conflicting changes that Git cannot automatically merge.

1. If two developers edited same line of code  
Git cannot understand which line to keep which creates a merge conflict
2. If a file is deleted and another person tried to modify it  
Git cannot automatically resolve this ,so developer has to decide whether to restore the file or keep it deleted

### **How to resolve them :**

1. Identify the root cause of the merge when we do git merge it will show us the merge conflicts .  
It will show something like below

```
<<<<<< HEAD
// recent changes
=====
// Changes from other branch
>>>>>> origin
```

2. Manually edit the conflicting files and make sure to remove the markers
3. Then add and commit the changes and merge the

### **Case Study 3: JDBC Library Management System**

**Question:** Develop a JDBC program to manage a library database. Include classes for managing books, borrowers, and transactions (such as issuing and returning books). Describe how you'd set up the database connection, query execution, and error handling.

ANS:

```
package assignment;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class databaseconnection {
    private static final String URL ="jdbc:mysql://localhost:3306/exam";
    private static final String USER ="root";
    private static final String PASSWORD="Rajitha@1175";

    public static Connection getConnection() throws SQLException{
        return DriverManager.getConnection(URL,USER,PASSWORD);
    }
}
```

```
package assignment;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class book {
    public static boolean updatequantity(int bookId, int newQuantity) {
        String query = "UPDATE Books SET quantity = ? WHERE bookid = ?";
        try(Connection conn=databaseconnection.getConnection();
            PreparedStatement ps=conn.prepareStatement(query))
        {
            ps.setInt(1, newQuantity);
            ps.setInt(2, bookId);

            int rowsAffected = ps.executeUpdate();
            return rowsAffected > 0;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }

    public book getBookById(int bookId) {
        String query = "SELECT * FROM Books WHERE bookid = ?";
```

```

public book getBookById(int bookId) {
    String query = "SELECT * FROM Books WHERE bookid = ?";

    try(Connection conn=databaseconnection.getConnection();
        PreparedStatement ps=conn.prepareStatement(query))
    {

        ps.setInt(1, bookId);
        ResultSet rs = ps.executeQuery();

        if (rs.next()) {
            System.out.println("bookid : "+rs.getInt("bookid"));
            System.out.println("booktitle :"+rs.getString("title"));
            System.out.println("bookquantity :"+rs.getInt("quantity"));

        }

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}

public int getBookquanById(int bookId) {
    String query = "SELECT * FROM Books WHERE bookid = ?";

    try(Connection conn=databaseconnection.getConnection();
        PreparedStatement ps=conn.prepareStatement(query))
    {

        ps.setInt(1, bookId);
        ResultSet rs = ps.executeQuery();

        if (rs.next()) {

            return rs.getInt("quantity");

        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return 0;
}
}

```

```

package assignment;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class borrowers {

    private int bId;
    private String name;

    public borrowers(int bId, String name) {
        this.bId=bId;
        this.name=name;
    }

    public int getBorrowerId() {
        return bId;
    }

    public String getName() {
        return name;
    }

    public static boolean addBorrower(int bId, String name) {
        String query = "INSERT INTO Borrowers VALUES (?, ?)";
        try(Connection conn=databaseconnection.getConnection();
            PreparedStatement ps=conn.prepareStatement(query))
        {

            ps.setInt(1, bId);
            ps.setString(2, name);
            int rowsAffected = ps.executeUpdate();
            return rowsAffected > 0;

        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }
}

```

```

package assignment;

import java.awt.print.Book;
import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class transactions {
    private int tId;
    private int bookId;
    private int bId;
    private Date issueDate;
    private Date returnDate;

    public static boolean issueBook(int tId, int bookId, int bId) {
        String query = "INSERT INTO transact VALUES (?, ?, ?, CURDATE(), null)";
        try (Connection conn = databaseconnection.getConnection();
            PreparedStatement ps = conn.prepareStatement(query))
        {
            ps.setInt(1, tId);
            ps.setInt(2, bookId);
            ps.setInt(3, bId);
            int rowsAffected = ps.executeUpdate();

            if (rowsAffected > 0) {
                book book = new book();

```

```

                    int quan = book.getBookquanById(bookId);
                    book.updatequantity(bookId, quan - 1);
                    return true;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }

    public static boolean returnBook(int tId) {
        Connection conn = null;
        PreparedStatement ps = null;

        String query = "SELECT bookId FROM Transact WHERE tId = ?";

        try {
            conn = databaseconnection.getConnection();
            ps = conn.prepareStatement(query);
            ps.setInt(1, tId);
            ResultSet rs = ps.executeQuery();

            if (rs.next()) {
                int bookId = rs.getInt("bookId");
                String query1 = "UPDATE Transact SET returnDate = CURDATE() WHERE tId = ?";

                ps = conn.prepareStatement(query1);
                ps.setInt(1, tId);

```

```

PreparedStatement ps = null;

String query = "SELECT bookId FROM Transact WHERE tId = ?";

try
{
    conn = databaseconnection.getConnection();
    ps = conn.prepareStatement(query);
    ps.setInt(1, tId);
    ResultSet rs = ps.executeQuery();

    if (rs.next()) {
        int bookId = rs.getInt("bookId");
        String query1 = "UPDATE Transact SET returnDate = CURDATE() WHERE tId

        ps = conn.prepareStatement(query1);
        ps.setInt(1, tId);
        int rowsAffected = ps.executeUpdate();

        if (rowsAffected > 0) {
            book book = new book();
            int quan=book.getBookquanById(bookId);
            book.updatequantity(bookId, quan + 1);
            return true;
        }
    }
} catch (SQLException e) {
    e.printStackTrace();
}
return false;
}
}

```

```

package assignment;

import java.awt.print.Book;

public class test {

    public static void main(String[] args) {

        book b=new book();
        System.out.println("current book count");
        b.getBookById(20);

        borrowers b1 = new borrowers(123,"rajitha");
        transactions t = new transactions();
        System.out.println("issuing book");
        t.issueBook(1100, 20, 123);
        System.out.println("book count after issuing");
        b.getBookById(20);
        System.out.println("book returned");
        t.returnBook(1100);
        b.getBookById(20);
    }
}

```

```
current book count
bookid : 20
booktitle :boxing
bookquantity :195
issuing book
book count after issuing
bookid : 20
booktitle :boxing
bookquantity :194
book returned
bookid : 20
booktitle :boxing
bookquantity :195
```

**Process:**

Initially after creating a project i added driver to it

Next created a database connection class to connect my java code to mysql database in this particular i provided username and credentials to access my mysql database

In book class two methods are created one to update the books and second to find a book by its book id.

In borrowers class required attributes are added

In transaction class issue and return methods are defined which describes the logic to update books inventory when a transaction is made

**Output:**

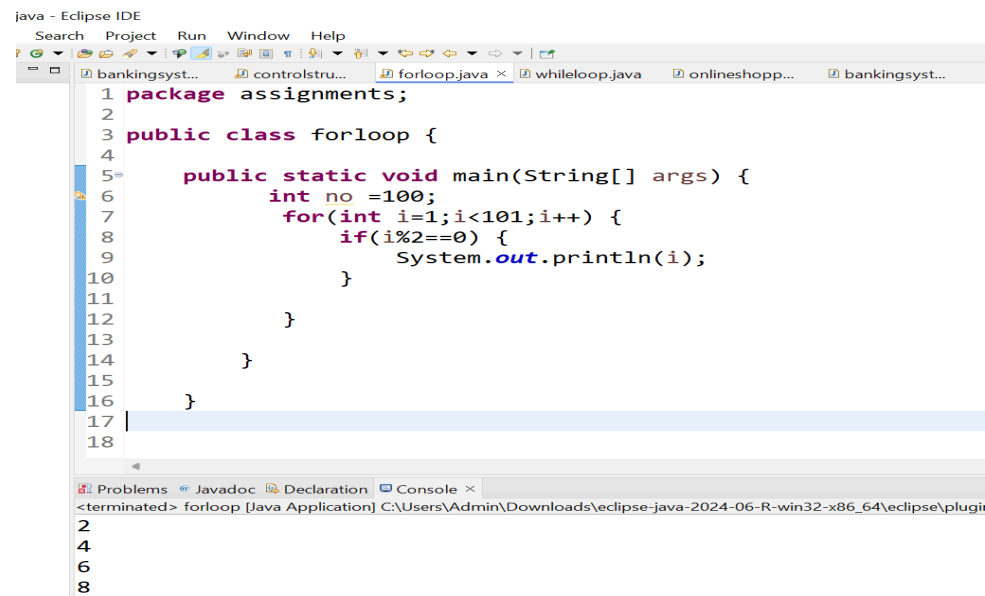
As seen in above output console after a transaction is made it should update the book quantity count in above example we can see initially the count is 195 when an issue transaction is made the count changed to 194 and again once the book is returned it changed to 195.

## Case Study 4: Control Structures in Java

**Question:** Write a Java program that uses for and while loops to print even numbers between 1 and 100. Explain the choice of control structure and discuss any potential edge cases.

ANS:

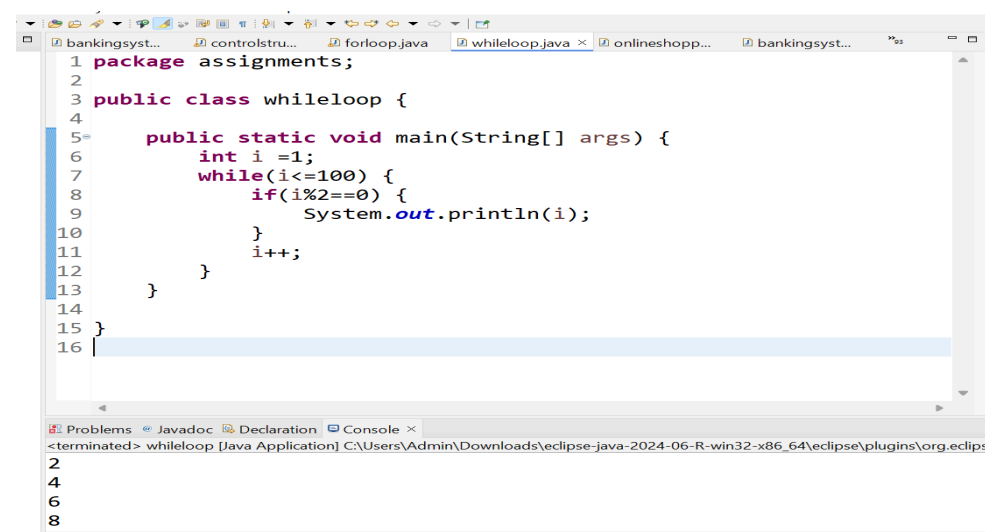
**Using for loop:** using for loop we know exact range of number we iterate and it provides a clear structure where the initialization, condition, and increment are handled in one place.



```
1 package assignments;
2
3 public class forloop {
4
5     public static void main(String[] args) {
6         int no = 100;
7         for(int i=1; i<101; i++) {
8             if(i%2==0) {
9                 System.out.println(i);
10            }
11        }
12    }
13
14 }
15
16 }
17
18 }
```

The screenshot shows the Eclipse IDE with a Java file named `forloop.java`. The code defines a package `assignments` and a public class `forloop`. Inside the `main` method, a variable `no` is set to 100. A `for` loop iterates from `i=1` to `i<101`. Inside the loop, an `if` statement checks if `i%2==0`, and if true, it prints `i` using `System.out.println(i)`. The console output shows the numbers 2, 4, 6, and 8.

**Using while loop:** this is more flexible and can be used when we don't know in advance how many times the loop will run, though in this case, it's equally valid to use it.



```
1 package assignments;
2
3 public class whileloop {
4
5     public static void main(String[] args) {
6         int i = 1;
7         while(i<=100) {
8             if(i%2==0) {
9                 System.out.println(i);
10            }
11            i++;
12        }
13    }
14 }
15
16 }
```

The screenshot shows the Eclipse IDE with a Java file named `whileloop.java`. The code defines a package `assignments` and a public class `whileloop`. Inside the `main` method, a variable `i` is initialized to 1. A `while` loop runs as long as `i<=100`. Inside the loop, an `if` statement checks if `i%2==0`, and if true, it prints `i` using `System.out.println(i)`. After the `if` block, `i` is incremented by 1 (`i++`). The console output shows the numbers 2, 4, 6, and 8.



## **Explanation of Choice of Control Structures:**

### **For loop:**

In this scenario for loop suites better because we know the number of iterations (i.e 1 to 100)  
In this program we have given fixed starting index and ending index and given a if condition to print only elements which are divisible by 2(i.e, even numbers).

### **While loop:**

Usually while loop is more flexible than a for loop and can be used when the number of iterations isn't immediately clear or when the increment logic is more complex.but in this case for loop suits better because we know the number of iterations

## **Edge Cases and Considerations:**

### **Starting Point:**

we start from 1 because we are given to find the even numbers between 1 and 100 if we had mistakenly given 0 as starting index it would unnecessarily print 0 also as it is divisible by 2

### **End Point:**

The loop should stop at 100, inclusive. If the loop condition is written incorrectly, such as using  $i < 100$  the program would stop at 98, missing the final even number (100).

### **Even Numbers Beyond 100:**

If the requirement were to print even numbers up to a different number (e.g. 200), we would need to adjust the loop's condition and range accordingly.

We're also assuming that the number 100 is included, which is important when dealing with inclusive bounds.

## **Case Study 5: Object-Oriented Programming - Online Shopping System**

**Question:** Create a design for an online shopping system using classes like Product, Cart, and Order. Define key attributes and methods for each class and describe how a user could add items to their cart and place an order.

**ANS:** used several concepts in this case study such as encapsulation, classes ,object etc

Below are the codes for each classes created

```

package assignments;
import java.util.ArrayList;
import java.util.List;

class products{

    private String id;
    private String name;
    private double price;

    public products(String id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

}

    public double getPrice() {
        return price;
    }

}

class Cart {
    private List<products> products;

    public Cart() {
        products = new ArrayList<>();
    }

    public void addProduct(products p) {
        products.add(p);
    }

    public void removeProduct(products p) {
        products.remove(p);
    }
}

```

```

    public List<products> viewProducts() {
        return products;
    }

    public double calculateTotal() {
        double total = 0.0;
        for (products p : products) {
            total += p.getPrice();
        }
        return total;
    }
}

class Order {
    private List<products> p;
    private double totalAmount;

    public Order(List<products> p, double totalAmount) {
        this.p = p;
        this.totalAmount = totalAmount;
    }

    public void ordersummary() {
        System.out.println("Order Details:");
        for (products p : p) {
            System.out.println("Product: " + p.getName() + ", Price: " + p.getPrice());
        }
        System.out.println("Total Amount: " + totalAmount);
    }
}

public class onlineshoppingsystem {

    public static void main(String[] args) {

        products p1=new products("10","mobile",11000);
        products p2=new products("20","laptop",40000);
        Cart c=new Cart();
        c.addProduct(p1);
        c.addProduct(p2);

        Order o=new Order(c.viewProducts(),c.calculateTotal());
        o.ordersummary();
    }
}

```

Problems @ Javadoc Declaration Console ×

<terminated> onlineshoppingsystem [Java Application] C:\Users\Admin\Downloads\eclipse-java-20

Order Details:  
 Product: mobile, Price: 11000.0  
 Product: laptop, Price: 40000.0  
 Total Amount: 51000.0

**Process:**

Create product, cart and order classes and initializes the attributes in each class

Create add, remove and view products in the cart class so that a product can be added or removed or viewed whenever a customer wants

Since a cart can have many number of products we store those products in list collection

Finally in order class we will calculate the total cost of the products in the cart and display the order summary

**Output:**

As seen in above picture of console output, initially we added products i.e, mobile and laptop later those products are added into cart then finally the products which are present in the cart are ordered which can be seen in order summary

**Case Study 6: Git Rollback and Recovery**

**Question:** A developer has made several commits and realized a mistake was introduced in one of the earlier commits. Explain how git revert and git reset can be used to undo changes, along with the advantages and risks of each approach.

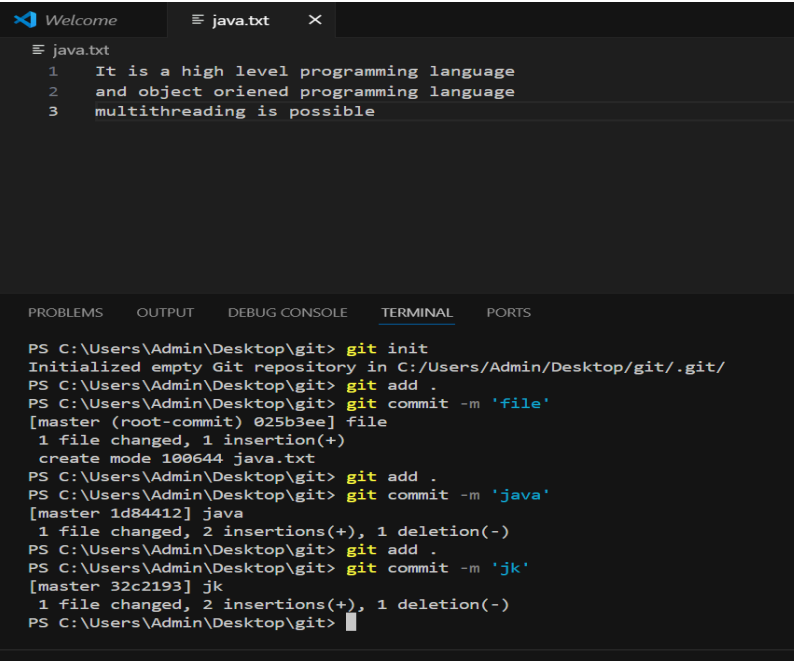
ANS:

There are several commands in git to revert the changes such as git revert and git reset

Git revert: command is used to undo a commit by creating a new commit that reverses the changes made by a previous commit.

**Process:**

1. Made changes to files and add and commit them



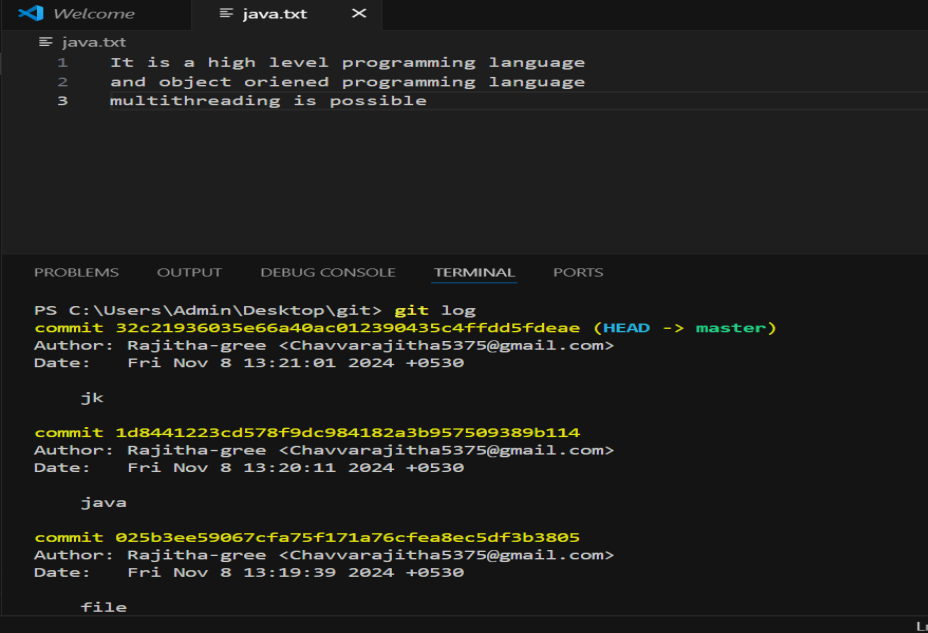
```

Welcome
java.txt
1  It is a high level programming language
2  and object oriented programming language
3  multithreading is possible

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Admin\Desktop\git> git init
Initialized empty Git repository in C:/Users/Admin/Desktop/git/.git/
PS C:\Users\Admin\Desktop\git> git add .
PS C:\Users\Admin\Desktop\git> git commit -m 'file'
[master (root-commit) 025b3ee] file
1 file changed, 1 insertion(+)
create mode 100644 java.txt
PS C:\Users\Admin\Desktop\git> git add .
PS C:\Users\Admin\Desktop\git> git commit -m 'java'
[master 1d84412] java
1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\Admin\Desktop\git> git add .
PS C:\Users\Admin\Desktop\git> git commit -m 'jk'
[master 32c2193] jk
1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\Admin\Desktop\git>
```

2) To perform git revert we need to find commit of particular commit which we want to revert that can be performed using git log as shown in below picture



```

Welcome
java.txt
1 It is a high level programming language
2 and object oriented programming language
3 multithreading is possible

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Admin\Desktop\git> git log
commit 32c21936035e66a40ac012390435c4ffdd5fdeae (HEAD -> master)
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:21:01 2024 +0530

    jk

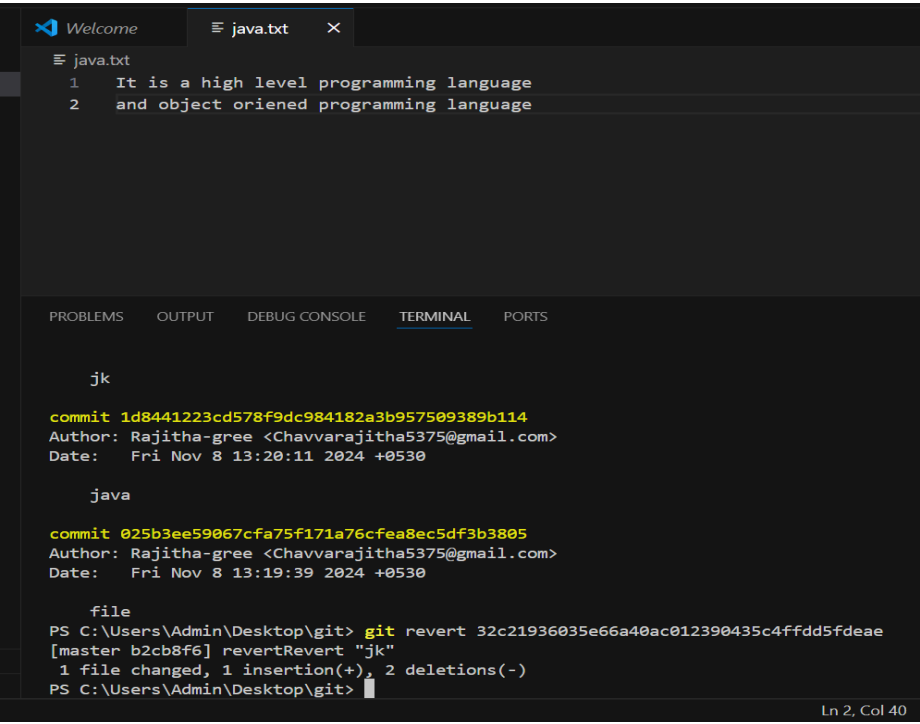
commit 1d8441223cd578f9dc984182a3b957509389b114
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:20:11 2024 +0530

    java

commit 025b3ee59067cfa75f171a76cfea8ec5df3b3805
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:19:39 2024 +0530

    file
Ln
```

3. Git revert commit id is the command to perform git revert as shown in below picture



```

Welcome
java.txt
1 It is a high level programming language
2 and object oriented programming language

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

jk

commit 1d8441223cd578f9dc984182a3b957509389b114
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:20:11 2024 +0530

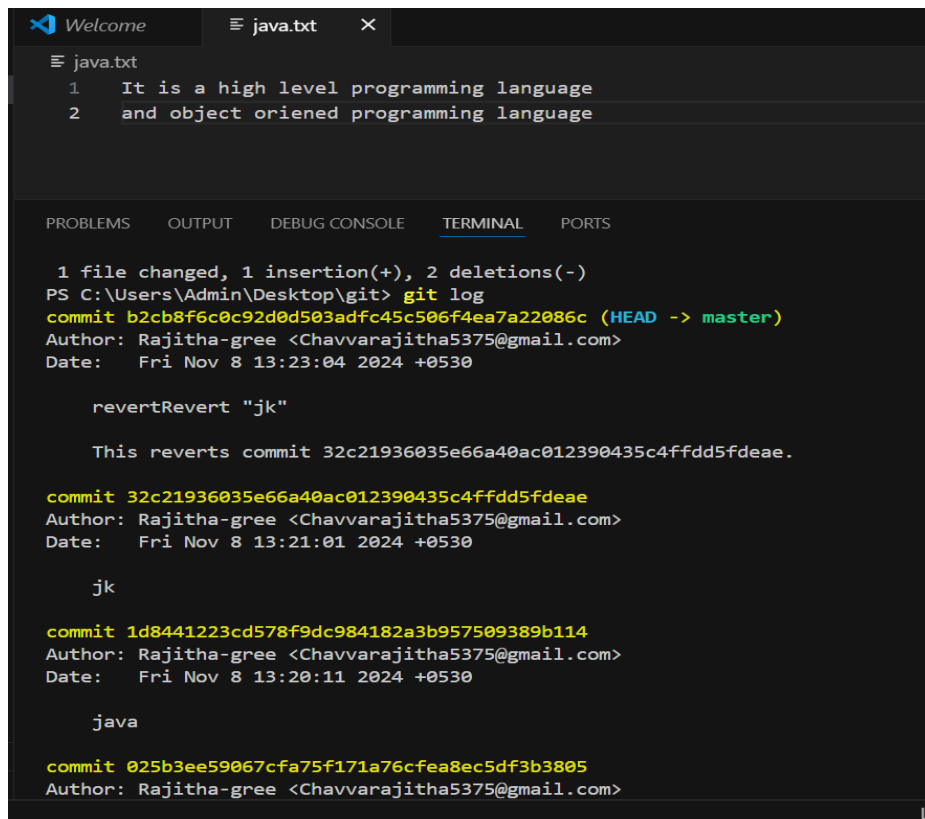
    java

commit 025b3ee59067cfa75f171a76cfea8ec5df3b3805
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:19:39 2024 +0530

    file

PS C:\Users\Admin\Desktop\git> git revert 32c21936035e66a40ac012390435c4ffdd5fdeae
[master b2cb8f6] revertRevert "jk"
1 file changed, 1 insertion(+), 2 deletions(-)
PS C:\Users\Admin\Desktop\git>
Ln 2, Col 40
```

4. We can see git revert command created a new commit to revert those changes instead of changing commit history .



```
Welcome  java.txt X
≡ java.txt
1 It is a high level programming language
2 and object oriented programming language

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

1 file changed, 1 insertion(+), 2 deletions(-)
PS C:\Users\Admin\Desktop\git> git log
commit b2cb8f6c0c92d0d503adfc45c506f4ea7a22086c (HEAD -> master)
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:23:04 2024 +0530

    revertRevert "jk"

    This reverts commit 32c21936035e66a40ac012390435c4ffdd5fdeae.

commit 32c21936035e66a40ac012390435c4ffdd5fdeae
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:21:01 2024 +0530

    jk

commit 1d8441223cd578f9dc984182a3b957509389b114
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:20:11 2024 +0530

    java

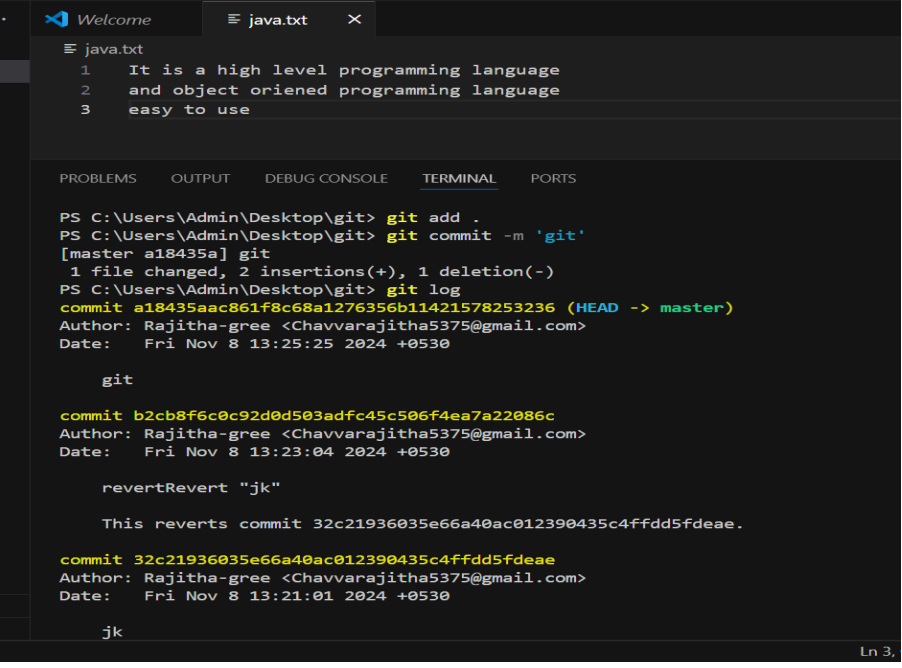
commit 025b3ee59067cfa75f171a76cfea8ec5df3b3805
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
```

**Git reset** :command is one of the most powerful and flexible commands in Git. It allows you to undo changes in your working directory, unstage changes, or even remove commits from history.it Modifies commit history by moving the HEAD to a specific commit, potentially erasing commits (depending on the option used).

There are several options in git reset depending on type of effects we require

#### Git soft reset :

1. Made changes and added and committed the changes



The screenshot shows the Visual Studio Code interface. At the top, there are two tabs: 'Welcome' and 'java.txt'. The 'java.txt' tab is active, displaying the following text:

```
1 It is a high level programming language
2 and object oriented programming language
3 easy to use
```

Below the editor, the 'TERMINAL' tab is active, showing the following commands and output:

```
PS C:\Users\Admin\Desktop\git> git add .
PS C:\Users\Admin\Desktop\git> git commit -m 'git'
[master a18435a] git
1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\Admin\Desktop\git> git log
commit a18435aac861f8c68a1276356b11421578253236 (HEAD -> master)
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:25:25 2024 +0530

    git

commit b2cb8f6c0c92d0d503adfc45c506f4ea7a22086c
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:23:04 2024 +0530

    revertRevert "jk"

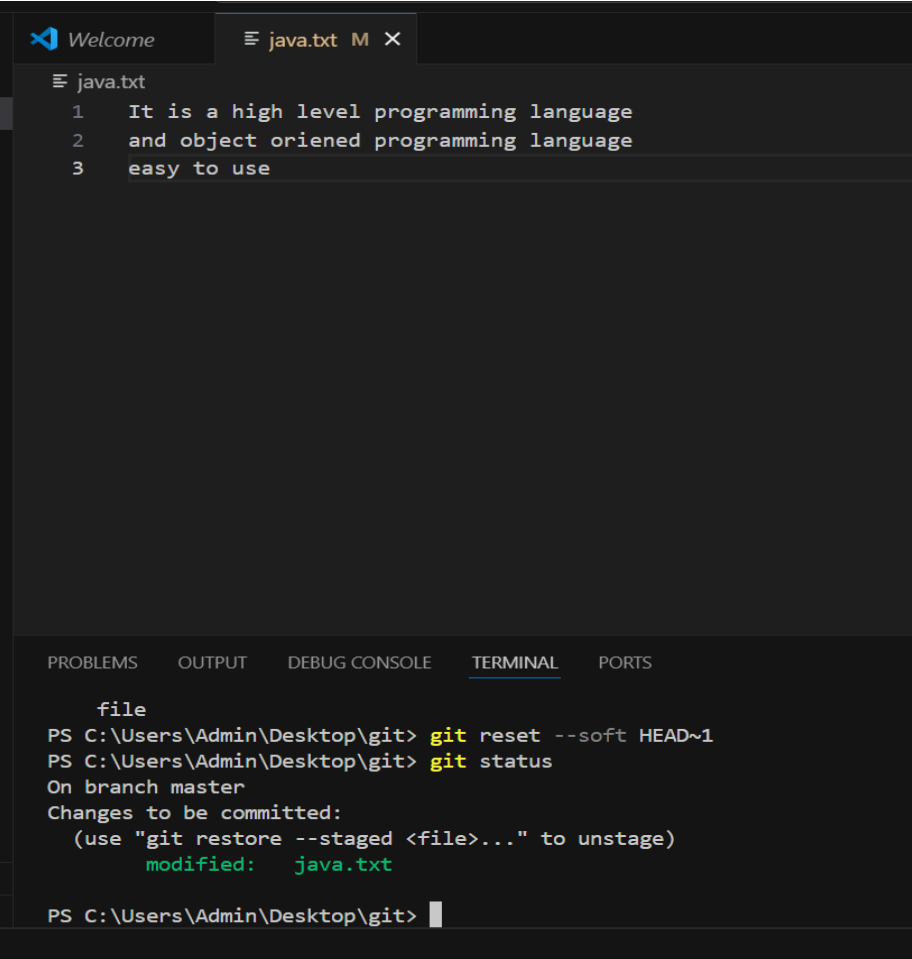
    This reverts commit 32c21936035e66a40ac012390435c4ffdd5fdeae.

commit 32c21936035e66a40ac012390435c4ffdd5fdeae
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:21:01 2024 +0530

    jk
```

The terminal output shows the execution of `git add .`, `git commit -m 'git'`, `git log`, and `git revert "jk"`. The `git log` command shows the commit history, including the current commit and the commit being reverted. The `git revert "jk"` command shows the commit being reverted and the commit being created to revert it.

2. Git reset --soft HEAD~1: this command Moves the HEAD to a specific commit but keeps changes staged for the next commit. As seen in below picture it keeps the changes in staging directory



The screenshot shows the Visual Studio Code interface. At the top, there are two tabs: 'Welcome' and 'java.txt M X'. The 'java.txt' tab is active, displaying the following text:

```
1 It is a high level programming language
2 and object oriented programming language
3 easy to use
```

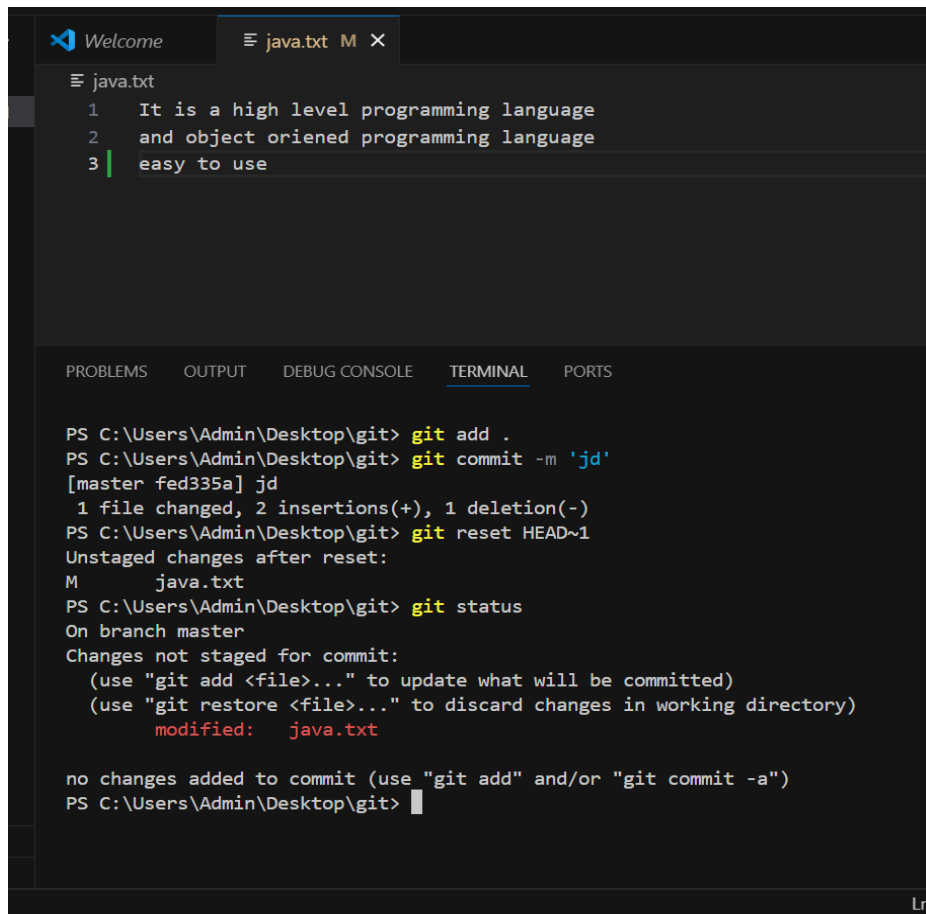
Below the editor, the 'TERMINAL' tab is active, showing the following commands and output:

```
file
PS C:\Users\Admin\Desktop\git> git reset --soft HEAD~1
PS C:\Users\Admin\Desktop\git> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   java.txt

PS C:\Users\Admin\Desktop\git>
```

The terminal output shows the execution of `git reset --soft HEAD~1` and `git status`. The `git status` command shows the current state of the repository, including the commit being reset to and the changes to be committed.

2. Git reset HEAD~1: Resets the HEAD and removes the file from staging area (default mode). Changes remain in your working directory. As seen in below picture it unstages the changes but keeps them in working directory



```
Welcome  java.txt M X

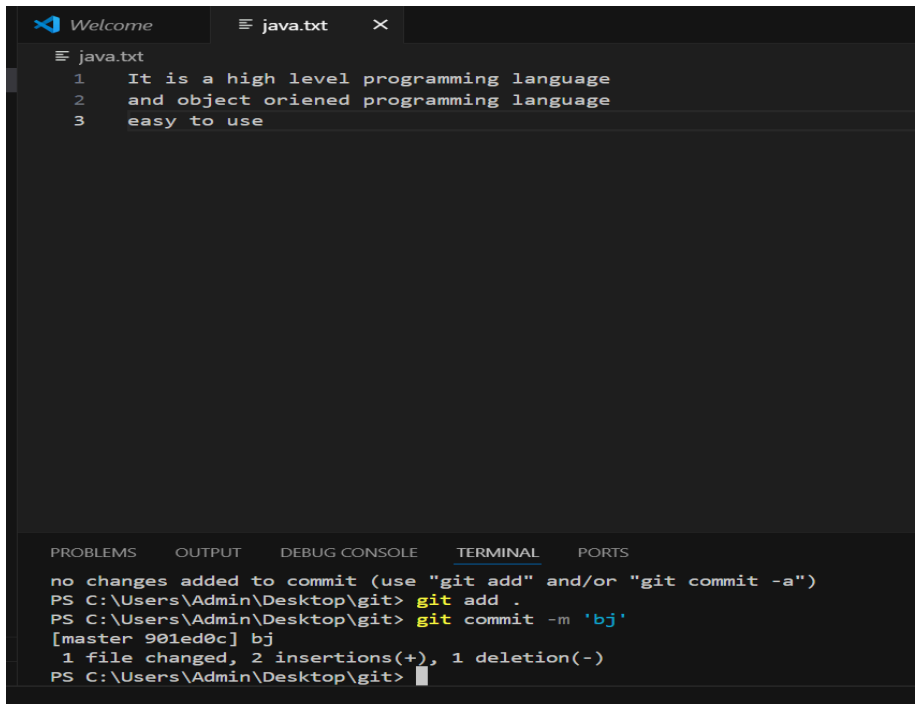
java.txt
1 It is a high level programming language
2 and object oriented programming language
3 easy to use

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Admin\Desktop\git> git add .
PS C:\Users\Admin\Desktop\git> git commit -m 'jd'
[master fed335a] jd
1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\Admin\Desktop\git> git reset HEAD~1
Unstaged changes after reset:
M    java.txt
PS C:\Users\Admin\Desktop\git> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   java.txt

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\Admin\Desktop\git>
```





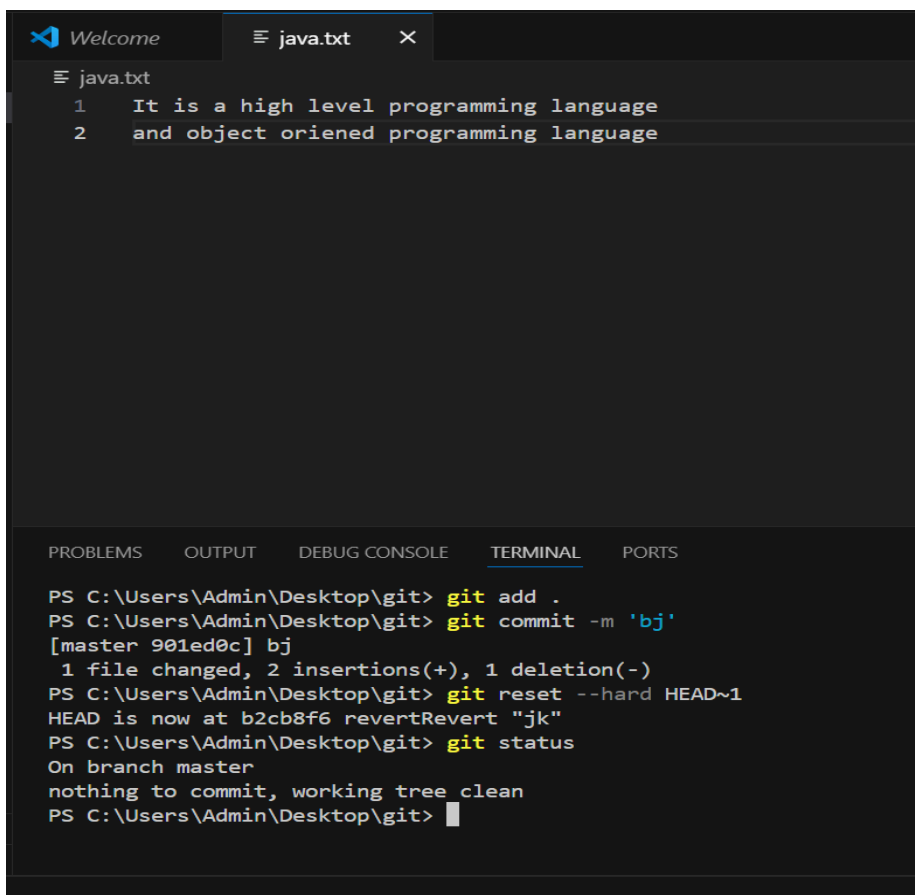
The screenshot shows the Visual Studio Code interface. The editor window displays a file named `java.txt` with the following content:

```
1 It is a high level programming language
2 and object oriented programming language
3 easy to use
```

The terminal window at the bottom shows the following commands and output:

```
no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\Admin\Desktop\git> git add .
PS C:\Users\Admin\Desktop\git> git commit -m 'bj'
[master 901ed0c] bj
 1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\Admin\Desktop\git>
```

Git reset --hard HEAD~1: Resets the HEAD, the staging area, and the working directory (discards all changes). As seen in below picture it deletes the changes from working directly

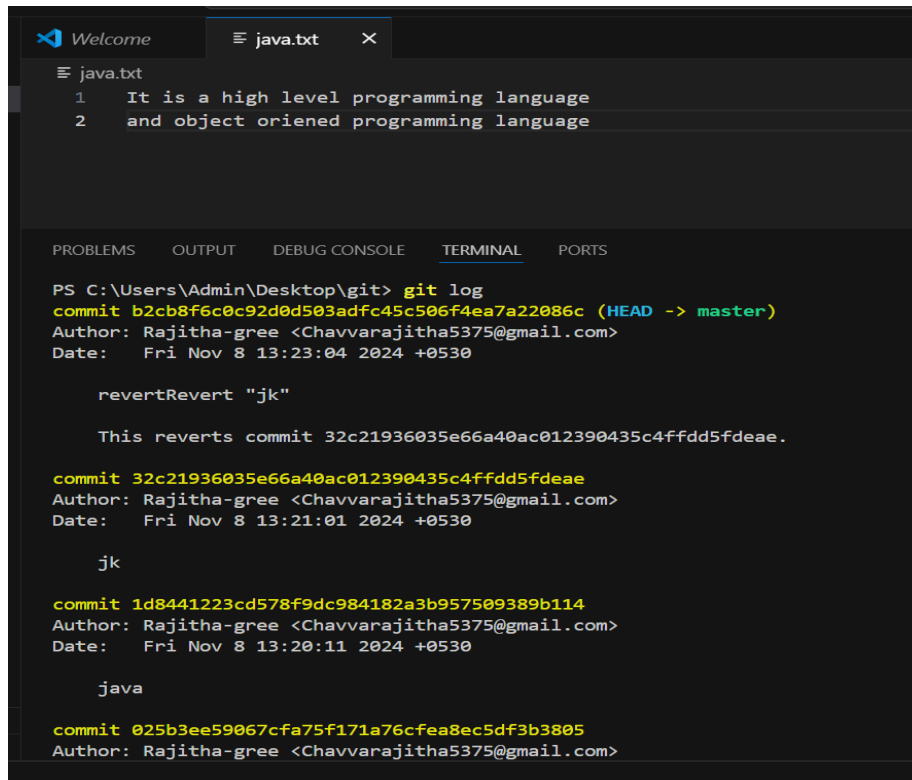


The screenshot shows the Visual Studio Code interface. The editor window displays the same file `java.txt` with the same content as the first screenshot:

```
1 It is a high level programming language
2 and object oriented programming language
```

The terminal window at the bottom shows the following commands and output:

```
PS C:\Users\Admin\Desktop\git> git add .
PS C:\Users\Admin\Desktop\git> git commit -m 'bj'
[master 901ed0c] bj
 1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\Admin\Desktop\git> git reset --hard HEAD~1
HEAD is now at b2cb8f6 revertRevert "jk"
PS C:\Users\Admin\Desktop\git> git status
On branch master
nothing to commit, working tree clean
PS C:\Users\Admin\Desktop\git>
```



```
Welcome
java.txt
1 It is a high level programming language
2 and object oriented programming language

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Admin\Desktop\git> git log
commit b2cb8f6c0c92d0d503adfc45c506f4ea7a22086c (HEAD -> master)
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:23:04 2024 +0530

    revertRevert "jdk"

    This reverts commit 32c21936035e66a40ac012390435c4ffdd5fdeae.

commit 32c21936035e66a40ac012390435c4ffdd5fdeae
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:21:01 2024 +0530

    jdk

commit 1d8441223cd578f9dc984182a3b957509389b114
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
Date: Fri Nov 8 13:20:11 2024 +0530

    java

commit 025b3ee59067cfa75f171a76cfea8ec5df3b3805
Author: Rajitha-gree <Chavvarajitha5375@gmail.com>
```

### Advantages of git revert:

- **keeps history:** The commit history remains intact, and all previous commits are preserved, which is important when collaborating with others.
- **Safe for shared branches:** Since it doesn't alter the commit history (it simply adds a new commit to reverse the changes), it's safe to use on branches that are shared with other developers.
- **Easy to use:** `git revert` automatically handles the creation of a commit and will open an editor to allow you to modify the commit message (if desired).

### Risks/Disadvantages of git revert:

- **Creates extra commits:** A new commit is created to undo the previous commit, which may clutter the commit history, especially if multiple reverts are required.
- **Possible conflicts:** If there are other changes in subsequent commits that conflict with the revert, Git will raise merge conflicts, which need to be manually resolved.

### Advantages of git reset:

**Powerful and flexible:** It can be used to completely remove commits from history (with `--hard`), or just reset the state of the working directory and staging area (with `--mixed` or `--soft`).

**Useful for local cleanup:** When you're working alone on a branch and want to tidy up the commit history (e.g., squashing commits or removing unwanted commits), git reset is very useful.

**Works well for early mistakes:** If the mistake was made in an early commit, git reset can be used to reset back to a clean state.

**Risks/Disadvantages of git reset:**

**Rewrites history:** If you have already pushed the commits to a shared repository, using git reset (especially with --hard) can cause problems for other developers. They may encounter conflicts when trying to pull or push their own changes.

**Potential data loss:** If you use --hard, any changes in the working directory or staging area will be lost permanently, which can be dangerous if not backed up or committed beforehand.

**Difficult to recover:** Once commits are reset and lost (with --hard), they are difficult to recover, unless you have backups or the commit hashes are still in the reflog.