

CCEE Mock- I | J2SE & OS

Total points 26/40 ?

I hope you all came prepared for this and going to take this test seriously. Consider this as your actual CCEE and don't fall in any of the malpractices because obviously this is for your preparation purpose only. Also, do analyse the concept where you lagged in this paper. In the end there is feedback field as well [Consider 1 min for that]

All the best. 😊

The respondent's email (**amolgavit158121@gmail.com**) was recorded on submission of this form.

0 of 0 points

PRN (12 Digits) *

250240320013



Centre *

Dropdown

Kharghar

Name *

Amol Gavit

Questions

26 of 40 points

J2SE & OS

✓ Objects are passed by value or reference? *

- ☒ 1. By value
- ☐ 2. By reference
- ☐ 3. It depends upon how you specify
- ☐ 4. None of the above

By value – In Java, primitive types (like int, double, char, etc.) are passed by value, meaning a copy of the actual value is passed to a method. If changes are made inside the method, they don't affect the original value in the calling code.

By reference – This might seem true at first glance, but Java does not support true pass-by-reference. Instead, objects are passed by value of their reference, meaning a method receives a copy of the reference (pointer) to the object, not the object itself. If the method modifies the object's internal fields, the change will reflect outside. However, reassigning the reference inside the method does not affect the original reference in the caller.

✗ Under which circumstances will a thread stop? *

0/1

- ☐ 1. the run() method that the thread stop
- ☐ 2. the call to the start() method of the thread is executing ends
- ☒ 3. the suspend() method is called on the Thread object
- ☐ 4. the wait() method is called on the Thread object

✗

Correct answer

- ☒ 1. the run() method that the thread stop

The run() method that the thread stop – This is correct because a thread stops when its run() method completes execution. Once the method finishes, the thread reaches the terminated state and cannot be restarted.

The call to the start() method of the thread is executing ends – Incorrect. The start() method only initiates the thread; it does not control when the thread stops.

The suspend() method is called on the Thread object – Incorrect. The suspend() method was deprecated in Java due to potential deadlocks and inconsistent behavior.

The wait() method is called on the Thread object – Incorrect. The wait() method only pauses a thread temporarily; it does not stop it permanently.



✗ How many child process will be created for the following code? *

0/1

```
void main() {  
    fork();  
    fork();  
    fork();  
}
```

Breakdown:

The first fork() creates 1 new process.

The second fork() doubles the existing processes, creating 2 more.

The third fork() doubles again, creating 4 more.

Thus, the total number of child processes is 7.

☐ 1. 6

☐ 2. 3

☒ 3. 8

☐ 4. 7

Correct answer

☒ 4. 7

✗

The fork() system call in C creates a new child process each time it is executed.

The number of child processes created follows the formula:

Total processes = 2^n

where n is the number of fork() calls.

In this case, there are 3 fork() calls:

$2^3 = 8$

(total processes)

However, this includes the original parent process, meaning the number of child processes is:

$8 - 1 = 7$

✓ If you write `System.exit(0)` at the end of try block, *
will the finally block still execute?

1/1

- ☐ 1. Yes
- ☒ 2. No
- ☐ 3. It depends upon return statement
- ☐ 4. Can't say



`System.exit(0)` immediately terminates the JVM, meaning no further code—including the finally block—will execute.

The finally block is typically guaranteed to run after a try block, but `System.exit(0)` halts execution before it gets the chance.

The only exception is if `System.exit(0)` throws a `SecurityException`, in which case the finally block might execute before termination

✗ What is the purpose of the following command sequence in Linux/Unix? * 0/1

`touch file.txt && cat file.txt | grep -E "[A-Z]{3}$"`

- ☐ 1. Creates a new empty file named file.txt and searches for a three-letter uppercase pattern at the beginning of each line within the file.
- ☐ 2. Appends the contents of file.txt to the existing file.txt and displays lines that match the regular expression "[A-Z]{3}\$".
- ☐ 3. Creates a new empty file named file.txt and searches for a three-letter uppercase pattern within the file, displaying the matching lines.
- ☒ 4. Copies the contents of file.txt to the standard input of the grep command and ✗ searches for lines that match the regular expression "[A-Z]{3}\$".

Correct answer

- ☒ 1. Creates a new empty file named file.txt and searches for a three-letter uppercase pattern at the beginning of each line within the file.

`touch file.txt` – Creates an empty file named file.txt if it does not already exist.

`cat file.txt | grep -E "[A-Z]{3}$"` – Reads the contents of file.txt and filters lines that match the regular expression `^[A-Z]{3}$`, which means:

The line must start (^) with exactly three uppercase letters ([A-Z]{3}).

The \$ ensures that the line contains only those three uppercase letters.

Correct Answer: Creates a new empty file named file.txt and searches for a three-letter uppercase pattern at the beginning of each line within the file.

Incorrect: The command does not append anything to file.txt; it only creates an empty file.

Incorrect: The command does search for the pattern, but since file.txt is empty, no matching lines will be found.

Incorrect: The command does not copy anything; it simply reads and filters the file contents.

✗ Which of the following scenarios could lead to a zombie process in Linux? *0/1

- ☐ 1. The parent process calls the wait() system call immediately after the child process exits.
- ☒ 2. The child process exits without the parent process explicitly waiting for it. ✗
- ☐ 3. The child process is terminated forcefully using the kill command.
- ☐ 4. The parent process is terminated before the child process finishes execution.

Correct answer

- ☒ 4. The parent process is terminated before the child process finishes execution.

The parent process calls the wait() system call immediately after the child process exits – Incorrect. If the parent calls wait(), the child process is properly cleaned up, preventing it from becoming a zombie.

The child process exits without the parent process explicitly waiting for it – Correct. If the parent does not call wait(), the child process remains in the process table as a zombie.

The child process is terminated forcefully using the kill command – Incorrect. Killing a process removes it from the system, preventing it from becoming a zombie.

The parent process is terminated before the child process finishes execution – Incorrect. If the parent dies before the child, the child becomes an orphan process, not a zombie. Orphan processes are adopted by the init process, which ensures they are properly cleaned up.

A zombie process occurs when a child process finishes execution but remains in the process table because its parent process has not yet read its exit status using the wait() system call. This happens because the operating system keeps the process entry to allow the parent to gather information about the terminated child.

✗ What will be the output of the following shell script? *

0/1

```
#!/bin/bash
```

```
files=$(ls *.txt)
```

ls *.txt – Lists all .txt files in the current directory.

```
count=$(echo "$files" | wc -w)
```

wc -w – Counts the number of words in the output of ls *.txt. Each filename is treated as a separate word.

```
echo "Total files: $count"
```

☐ 1. Total files: 0

☒ 2. Total files: 1

echo "\$files" – If no .txt files exist, ls *.txt returns an error, and \$files remains empty. ✗

☐ 3. Total files: 2

☐ 4. Syntax error

count=\$(echo "\$files" | wc -w) – Since \$files is empty, wc -w returns 0.

Correct answer

☒ 1. Total files: 0

Why "Total files: 0"?

If there are no .txt files, \$files is empty, and wc -w counts 0 words.

If there were .txt files, the output would depend on the number of filenames.

- ✓ If you will run following code what will be the * result?

```
public class RTExcept {  
    public static void throwit () {  
        System.out.print("throw it ");  
        throw new RuntimeException();  
    }  
    public static void main(String [] args) {  
        try {  
            System.out.print("hello ");  
            throwit();  
        }  
        catch (Exception re ) {  
            System.out.print("caught ");  
        }  
        finally {  
            System.out.print("finally ");  
        }  
        System.out.println("after ");  
    }  
}
```

- ☒ 1. hello throw it caught finally after
- ☐ 2. hello throw it RuntimeException caught after
- ☐ 3. Compilation fails

Execution starts in main():

"hello " is printed.

throwit() is called.

Inside throwit():

"throw it " is printed.

A RuntimeException is thrown.

Handling the Exception:

The exception is caught in the catch block.

"caught " is printed.

Finally Block Execution:

"finally " is printed.

After Finally Block:

"after " is printed.

The program does not terminate inside the catch block, so execution continues.

☐ 4. hello throw it caught finally after RuntimeException

✓ What is the error in the following code? *

1/1

```
class Test {abstract void display( );}
```

- ☐ 1. No error
- ☐ 2. Method display() should be declared as static
- ☒ 3. Test class should be declared as abstract
- ☐ 4. Test class should be declared as public



Explanation:

The error in the given code is that the class Test contains an abstract method (display()), but the class itself is not declared as abstract. In Java:

A class must be declared as abstract if it contains at least one abstract method.

Abstract methods do not have a body and must be implemented by subclasses.

No error – Incorrect. There is an error because the class is missing the abstract keyword.

Method display() should be declared as static – Incorrect. Abstract methods cannot be static because they are meant to be overridden in subclasses.

Test class should be declared as abstract – Correct. The class must be declared as abstract to allow abstract methods.

Test class should be declared as public – Incorrect. The visibility (public) is unrelated to the error.



Consider the following code in file Sample.java

```
public class Sample implements IInt
{
    public static void main(String[] args){
        Sample s = new Sample(); //1
        int j = s.thevalue; //2
        int k = IInt.thevalue; //3
        int l = thevalue; //4
    }
}
interface IInt
{
    int thevalue = 0;
}
```

What will happen when the above code is compiled and run?

- 1. It will give an error at compile time at line //1
- 2. It will give an error at compile time at line //2
- 3. It will give an error at compile time at line //3
- 4. It will compile and run without any problem.

4. It will compile and run without any problem.

The given code defines:

An interface IInt with a constant thevalue = 0.

A class Sample that implements IInt.

Breakdown of Each Line:

Line //1:

java

Sample s = new Sample();

This correctly creates an instance of Sample. No error here.

Line //2:

java

int j = s.thevalue;

Since Sample implements IInt, it inherits thevalue.

Interface variables are implicitly public, static, and final, meaning thevalue is accessible via an instance (s.thevalue), though it's not recommended.

Line //3:

java

int k = IInt.thevalue;

Since thevalue is static, it should be accessed via the interface name (IInt.thevalue).

This is correct and preferred.

Line //4:

java

int l = thevalue;

Since thevalue is static, it can be accessed without an instance inside Sample.

✓ Suppose you create a class Cylinder to be a subclass of Circle. Analyze the following code: *

```
class Cylinder extends Circle{
    double length;
    Cylinder(double radius){
        Circle(radius);
    }
}
```

1/1

- ☐ 1. The program compiles fine, but you cannot create an instance of Cylinder because the constructor does not specify the length of the cylinder.
- ☒ 2. The program has a syntax error because you attempted to invoke the Circle class's constructor illegally. ✓
- ☐ 3. The program compiles fine, but it has a runtime error because of invoking the Circle class's constructor illegally.
- ☐ 4. None of the above

Explanation:

The issue in the given code is in the **Cylinder constructor**:

```
java
Cylinder(double radius){
    Circle(radius);
}
```

The statement `Circle(radius);` is incorrect because it attempts to call the Circle constructor without using `super()`.

In Java, when a subclass calls a constructor of its superclass, it must use `super(arguments)`, not just the superclass name.

Why Does This Cause a Syntax Error?

The compiler expects a valid constructor call to the superclass.

Since `Circle(radius);` is not a valid way to invoke a superclass constructor, the program fails to compile.

Corrected Code:

```
java
class Cylinder extends Circle {
    double length;

    Cylinder(double radius) {
        super(radius); // Correct way to call the Circle constructor
    }
}
```

Steps to Create a Daemon

Process:

Fork the process – This creates a child process.

Terminate the parent process – The child process becomes orphaned and is adopted by init.

Call setsid() – This creates a new session and detaches the process from any terminal.

Change the working directory – Typically set to / to avoid locking a specific filesystem.

Read (r) → 4

Write (w) → 2

Execute (x) →

✓ How can a daemon process be created in Unix-like systems? *

- ☐ 1. By using a specific system call like "create_daemon()".
- ☐ 2. By running a command with the "--daemon" flag in the terminal.
- ☐ 3. By modifying the user's shell profile to start the process at login.
- ☒ 4. By forking an existing process, closing standard file descriptors, and changing the working directory.

Breakdown of Each Option:

Incorrect: There is no system call named "create_daemon()" in Unix/Linux.

Incorrect: Some commands support a --daemon flag, but this does not create a true daemon process.

Incorrect: Modifying the shell profile starts a process at login but does not make it a daemon.

Correct: The standard method for creating a daemon involves forking, detaching, and closing file descriptors

✓ In Linux/Unix systems, which command is used to change the permissions of a file to allow read, write, and execute access for the owner and read-only access for the group and others?

*1/1

- ☐ 1. chmod 664 file.txt
- ☒ 2. chmod 744 file.txt
- ☐ 3. chmod 777 file.txt
- ☐ 4. chmod 555 file.txt

Explanation:
The chmod command in Linux/Unix is used to modify file permissions. The numeric mode 744 sets the following permissions:

Owner (User): Read (r), Write (w), Execute (x) → Full access

Group: Read (r) → Can only read

Others: Read (r) → Can only read



✓ Which of the following is not a keyword? *

1/1

- ☒ 1. main
- ☐ 2. public
- ☐ 3. static
- ☐ 4. void



Breakdown of Each Option:

main – Not a keyword.

main is just a method name, specifically the entry point for Java programs (`public static void main(String[] args)`).

It is not a reserved keyword in Java.

public – Keyword.

public is an access modifier that defines visibility.

static – Keyword.

static is used to define class-level members that do not require an instance.

void – Keyword.



void is a return type modifier, indicating that a method does not return a value.

✗ Which scheduling algorithm is most suitable for real-time systems where meeting strict deadlines is a critical requirement? *0/1

- ☐ 1. First-Come, First-Served (FCFS) scheduling
- ☐ 2. Round Robin (RR) scheduling
- ☒ 3. Priority scheduling
- ☐ 4. None of the above

Correct answer

- ☒ 4. None of the above

Explanation:

Real-time systems require scheduling algorithms that ensure tasks meet strict deadlines. The most suitable scheduling algorithm for such systems is Earliest Deadline First (EDF) scheduling, which is not listed among the options.

Breakdown of Each Option:

First-Come, First-Served (FCFS) scheduling – Incorrect. FCFS executes processes in the order they arrive, without considering deadlines. It is not suitable for real-time systems where timing constraints are critical.

Round Robin (RR) scheduling – Incorrect. RR assigns equal time slices to processes, making it fair but not deadline-sensitive. It is better for time-sharing systems, not real-time applications.

Priority scheduling – Incorrect. While priority scheduling considers process importance, it does not guarantee deadline adherence. It can lead to priority inversion, where high-priority tasks are delayed.

None of the above – Correct. The best scheduling algorithm for real-time systems is Earliest Deadline First (EDF).

EDF schedules tasks based on their deadlines, ensuring critical tasks complete on time.

Explanation:

The issue in the given code is that the static initialization block is trying to modify a non-static variable (x), which is not allowed in Java.

Breakdown of the Code:

java

```
public class Test {  
    int x; // Instance variable  
    static { x++; } // Error:  
    Cannot access non-static  
    variable in a static block  
}
```

x is an instance variable, meaning it belongs to each object of the class.

Static blocks are executed once when the class is loaded, before any object is created.

Since x is not static, it does not exist when the static block runs, causing a compilation error.

✓ Analyze the following code: *

1/1

```
public class Test{  
    int x;  
    static {x++;}  
}
```

- ☐ 1. The program cannot be compiled, because the statement x++ must be placed inside a method or a constructor.
- ☐ 2. When you construct an instance of Test, the value of x becomes 0.
- ☒ 3. The program cannot be compiled, because x is non-static, but is used in a static initialization block. ✓
- ☐ 4. When you construct an instance of Test, the value of x becomes 1.

Breakdown of Each Option:

Incorrect: The error is not about needing a method or constructor; it's about accessing a non-static variable in a static block.

Incorrect: The program does not compile, so x never gets assigned a value.

Correct: The error occurs because x is non-static, but the static block tries to modify it.

Incorrect: The program does not compile, so x never gets assigned 1.

Key Concept:

The finalize() method is called by the Garbage Collector before an object is destroyed.

It is not a reserved keyword but a method that can be overridden for cleanup operations.

However, manual use of finalize() is discouraged in modern Java programming due to unpredictable execution timing.

✗ Because finalize() belongs to the java.lang.Object, it is present * in all _____

- ☒ 1. objects
- ☐ 2. classes
- ☐ 3. methods
- ☐ 4. none of above

Correct answer

- ☒ 2. classes

Explanation:

The finalize() method is part of java.lang.Object, which is the root class for all Java classes.

Since every class in Java implicitly extends Object, the finalize() method is inherited by all classes.

Breakdown of Each Option:

Objects – Incorrect. While every object can have a finalize() method, the method itself is defined at the class level, not per object.

Classes – superclass of all Java classes, finalize() is present in all classes.

Methods – Incorrect. finalize() is a method, but the question asks where it is present. It is not present in all methods.

None of the above – Incorrect. The method is inherited by all classes, making this option invalid.

✓ Sharing of CPU time is called as Time slicing or Time sharing. *

- ☒ 1.True
- ☐ 2.False
- ☐ 3.Time extra dilwado plz
- ☐ 4.CPU se share krwane ki responsibility phr meri

Explanation:

Time slicing and time sharing are both techniques used in operating systems to manage CPU time among multiple processes.

Time slicing refers to dividing CPU time into small intervals (called time slices or quanta) and allocating them to processes in a round-robin fashion.

Time sharing is a broader concept where multiple users or processes share CPU resources efficiently, ensuring fair execution.

Explanation:
In Java, interface variables have strict rules:

They are implicitly public, static, and final.

They must be initialized at the time of declaration.

They cannot be protected or transient.

✓ Which is valid declaration within an interface? *

- ☒ 1. public static short stop = 23
- ☐ 2. protected short stop = 23
- ☐ 3. transient short stop = 23;
- ☐ 4. final void madness(short stop);

Breakdown of Each Option:

public static short stop = 23; – Correct
This follows Java’s interface rules:
public → Accessible everywhere.
static → Belongs to the interface, not instances.
final (implicitly) → Cannot be changed after initialization.

protected short stop = 23; – Incorrect
Interface variables cannot be protected because interfaces do not support access control beyond public.

transient short stop = 23; – Incorrect
Interface variables cannot be transient, as transient is used for serialization, which does not apply to interface constants.

final void madness(short stop); – Incorrect
Interface methods cannot be final, because they are meant to be overridden by implementing classes.

Which system call is used to clean up zombie processes in Linux? *

- ☒ 1. wait()
- ☐ 2. exit()
- ☐ 3. fork()
- ☐ 4. exec()

Breakdown of Each Option:

wait() – Correct
The parent process calls wait() to collect the exit status of a terminated child, preventing zombie processes.
If wait() is not called, the child remains in the process table as a zombie.

exit() – Incorrect
exit() is used by a process to terminate itself, but it does not clean up zombie processes.

fork() – Incorrect
fork() creates a new child process but does not handle zombie cleanup.

exec() – Incorrect
exec() replaces the current process image but does not affect zombie processes.

Explanation:

A zombie process occurs when a child process terminates but remains in the process table because its parent has not yet read its exit status. The wait() system call is used by the parent process to clean up zombie processes by retrieving their exit status.

Explanation:
The issue in the given code is the line:

java
s.reverse();
String does not have a reverse() method, so this line causes a compilation error.

StringBuffer does have a reverse() method, which correctly reverses its contents.

✓ What will be the result of attempting to compile and run the following program? *

1/1

```
public class TestClass
{
    public static void main(String args[] ){
        String s = "hello";
        StringBuffer sb = new StringBuffer("hello");
        sb.reverse();
        s.reverse();
        if(s == sb.toString()) System.out.println("Equal");
        else
            System.out.println("Not Equal");
    }
}
```

- ☐ 1. It will print 'Equal'
- ☐ 2. It will print 'Not Equal'
- ☒ 3. Compilation error as there is no reverse() method in class String
- ☐ 4. Runtime error



Breakdown of Each Option:
It will print 'Equal' – Incorrect. The code does not compile, so no output is printed.
It will print 'Not Equal' – Incorrect. The code does not compile, so this comparison never happens.
Compilation error as there is no reverse() method in class String – Correct. String is immutable and does not support direct reversal.
Runtime error – Incorrect. The error occurs at compile time, not runtime.

Explanation:

The issue in the given code is the line:

java

Boolean b = (x = y);

= is an assignment operator, not a comparison operator.

The assignment x = y tries to store a double value (100.1) into an int variable (x), which is not allowed without explicit type casting.

Additionally, Boolean cannot store an integer or double value, leading to a compilation error.

✓ What will be the result ? *

```
class Equals{  
    public static void main(String[] args){  
        int x= 100;  
        double y = 100.1;  
        Boolean b = (x=y);  
        System.out.println(b);  
    }  
}
```

- ☐ 1. true
- ☐ 2. false
- ☒ 3. Compilation fails
- ☐ 4. An exception is thrown at runtime

Breakdown of Each Option:

true – Incorrect. The code does not compile, so no output is produced.

false – Incorrect. The code does not compile, so no comparison happens.

Compilation fails – Correct. The error occurs due to invalid assignment and type mismatch.

An exception is thrown at runtime – Incorrect. The error occurs at compile time, not runtime.



✗ Which scheduling algorithm is most suitable for minimizing the average turnaround time and waiting time in a system where all processes have the same priority? *0/1

- ☐ 1. First-Come, First-Served (FCFS) scheduling
- ☒ 2. Round Robin (RR) scheduling
- ☐ 3. Shortest Job Next (SJN) scheduling
- ☐ 4. Shortest Remaining Time (SRT) scheduling

Correct answer

- ☒ 3. Shortest Job Next (SJN) scheduling

SJN scheduling minimizes average waiting time because shorter processes finish quickly, allowing longer processes to start sooner. ✗

It works best in non-preemptive environments where all processes have the same priority.

Breakdown of Each Option:

First-Come, First-Served (FCFS) scheduling – Not optimal
Processes are executed in the order they arrive, which can lead to long waiting times if a short process arrives after a long one.

Round Robin (RR) scheduling – Not optimal
RR assigns equal time slices to processes, making it fair but not efficient for minimizing turnaround time.

Shortest Job Next (SJN) scheduling – Correct
By prioritizing shorter processes, it minimizes waiting time and turnaround time.

Shortest Remaining Time (SRT) scheduling – Not optimal
SRT is a preemptive version of SJN, but it requires continuous recalculations, making it less predictable.

Explanation:

Shortest Job Next (SJN), also known as Shortest Job First (SJF), is the most efficient scheduling algorithm for minimizing average turnaround time and waiting time when all processes have the same priority.

It selects the process with the smallest burst time first, ensuring that shorter tasks complete quickly, reducing overall waiting time.



Explanation:
An orphan process is a child process whose parent has terminated before the child finishes execution.

In Linux/Unix, orphan processes are automatically adopted by the init process (PID 1), which becomes their new parent.

The init process ensures that orphan processes continue running properly and are eventually cleaned up when they terminate

Explanation:
Hashtable is the only collection class in the given options that provides built-in synchronization.

It allows key-value associations, meaning elements are stored and accessed using unique keys.

Unlike HashMap, which is not synchronized, Hashtable ensures thread safety, making it suitable for multi-threaded environments.

✗ What is the role of the init process in handling orphan processes in Linux? *0/1

- ☐ 1. To immediately terminate and remove orphan processes from the process table.
- ☐ 2. To assign a new parent process to orphaned processes.
- ☒ 3. To collect the exit status of orphan processes and release their system resources. ✗
- ☐ 4. To adopt and become the new parent process of orphan processes.

Correct answer

- ☒ 4. To adopt and become the new parent process of orphan processes.

Breakdown of Each Option:

SortedMap – Incorrect.

SortedMap is an interface, not a concrete class.

It does not provide synchronization.

TreeMap – Incorrect.

TreeMap maintains sorted order of keys but is not synchronized.

It requires external synchronization for thread safety.

TreeSet – Incorrect.

TreeSet is a set, not a map, meaning it does not store key-value pairs.

It also lacks built-in synchronization. ✓

Hashtable – Correct.

It provides synchronized methods, ensuring thread safety.

It stores key-value pairs, making it suitable for concurrent access.

✓ Which collection class allows you to access its elements by associating a key with an element's value, and provides synchronization? *

- ☐ 1. java.util.SortedMap
- ☐ 2. java.util.TreeMap
- ☐ 3. java.util.TreeSet
- ☒ 4. java.util.Hashtable

Explanation:

The chown command in Linux/Unix is used to change the ownership of a file or directory.

It can also be used to assign a new owner to a device node, making it the correct choice.

✗ Which command in Linux/Unix systems is used to change the ownership of a file or directory, but can also be used to assign a new owner to a device node? *0/1

☐ 1. chown

☒ 2. chmod

☐ 3. chgrp

☐ 4. ls

✗

Correct answer

☒ 1. chown

Breakdown of Each Option:

chown – Correct

Used to change the owner of a file, directory, or device node.

Syntax: chown new_owner file_name

Example: chown user1 file.txt (Changes ownership to user1)

chmod – Incorrect

Used to change file permissions, not ownership.

Example: chmod 755 file.txt (Changes permissions)

chgrp – Incorrect

Used to change the group ownership, not the file owner.

Example: chgrp group1 file.txt (Changes group ownership)

ls –(?) Incorrect

Used to list files and directories, but does not modify ownership.

Explanation:

In Java, a class cannot be both abstract and final at the same time.

Abstract classes are meant to be extended by subclasses, requiring implementation of abstract methods.

Final classes cannot be extended, meaning no subclass can inherit from them.

Explanation:

The cp command in Linux/Unix is used to copy files or directories. The syntax:

bash

cp file1.txt file2.txt

Copies file1.txt to file2.txt.

If file2.txt already exists, it will be overwritten.

If file2.txt does not exist, it will be created.



✗ A class may be both abstract and final. *

- ☒ 1.True
- ☐ 2.False
- ☐ 3.Class ki marzi
- ☐ 4.Mein kisiki life m taang nhin adhata

Correct answer

- ☒ 2.False



Why is this False?

If a class is abstract, it is designed to be inherited.

If a class is final, it cannot be inherited.

Since these two concepts contradict each other, Java does not allow a class to be both abstract and final

✓ What does the following command do in Linux/Unix? *

cp file1.txt file2.txt

- ☒ 1. Copies file1.txt to file2.txt and overwrites file2.txt if it already exists.
- ☐ 2. Renames file1.txt to file2.txt.
- ☐ 3. Moves file1.txt to a directory named file2.txt.
- ☐ 4. Copies file1.txt and creates a new empty file named file2.txt.

Breakdown of Each Option:

Copies file1.txt to file2.txt and overwrites file2.txt if it already exists – This is the expected behavior of cp.

Renames file1.txt to file2.txt – Incorrect
Renaming is done using the mv command, not cp.

Moves file1.txt to a directory named file2.txt – Incorrect
Moving files is done using mv, not cp.

Copies file1.txt and creates a new empty file named file2.txt – Incorrect
cp copies the contents of file1.txt to file2.txt, it does not create an empty file.

Explanation:

A daemon process is a background process that runs independently of user interaction and provides essential system services. These processes are typically started at system boot and continue running until the system shuts down.

✗ What is the purpose of a daemon process? *

0/1

- ☐ 1. To provide a user-friendly interface for interacting with the operating system.
- ☐ 2. To perform background tasks and provide services to other processes or users.
- ☒ 3. To allocate system resources and manage memory for running applications. ✗
- ☐ 4. To prioritize and schedule processes for efficient utilization of system resources.

Correct answer

- ☒ 2. To perform background tasks and provide services to other processes or users.

Key Characteristics of Daemon Processes:
Runs in the background without direct user control.

Provides system services such as logging, network management, and scheduling.

Detached from the terminal, meaning they do not require user input.

Often adopted by the init process (PID 1) after their parent terminates.

Breakdown of Each Option:

To provide a user-friendly interface for interacting with the operating system – Incorrect.

Daemons do not provide direct user interfaces; they work silently in the background.

To perform background tasks and provide services to other processes or users – Correct.

Daemons handle system-level tasks like network monitoring, printing, and scheduling.

To allocate system resources and manage memory for running applications – Incorrect.

Resource allocation is managed by the kernel, not daemon processes.

To prioritize and schedule processes for efficient utilization of system resources – Incorrect.

Process scheduling is handled by the scheduler, not daemons.





*

1/1

4. What will be the output of the following code?

```
public class exception_demo
{
    public static void main(String str[]){
        int i=1, j=1;
        try
        {
            i++;
            j--;
            if(i/j > 1)
            i++;
        }
        catch(Exception e)
        { System.out.println("Exception"); }
        catch(ArithmeticException e)
        { System.out.println("arithmetic exception"); }
        catch(ArrayIndexOutOfBoundsException e)
        { System.out.println("Array index exception"); }
        finally
        { System.out.println("finally"); }
        System.out.println("after exceptions ");
    }
}
```

- ☒ 1. Give compilation error
- ☐ 2. arithmetic exception

1. Throwable (Root Class)

The base class for all exceptions and errors.

2. Exception (Handles recoverable errors)

Used for conditions that applications should handle.

Includes both checked and unchecked exceptions.

Checked Exceptions (Must be handled or declared)

IOException → File handling errors.

SQLException → Database access errors.

ClassNotFoundException → Missing class definitions.

Unchecked Exceptions (Runtime exceptions)

RuntimeException → Base class for runtime errors.

NullPointerException → Accessing an object reference that is null.

ArrayIndexOutOfBoundsException → Accessing an invalid array index.

ArithmeticException → Division by zero.

IllegalArgumentException → Invalid method arguments.

3. Error (Handles system-level failures)

Used for critical failures that applications should not handle.

Examples:

OutOfMemoryError → Insufficient memory.

StackOverflowError → Excessive recursion.

VirtualMachineError → JVM-related failures.

Why Does This Cause a Compilation Error?

Exception hierarchy is not correctly maintained:

ArithmeticException and ArrayIndexOutOfBoundsException are specific exceptions.

Exception is a parent class for these exceptions.

More specific exceptions must be caught first, followed by broader ones.

Since Exception is broader and placed first, later catch blocks become unreachable, causing a compilation error.



Explanation:

In Java, object references are used to store memory addresses of objects.

To explicitly drop an object reference, you can assign it to null, which means the reference no longer points to any object.

Once a reference is set to null, the object it previously pointed to becomes eligible for garbage collection, freeing up memory.

- ☐ 3. arithmetic exception finally
- ☐ 4. None of the above

✓ You can explicitly drop a object reference by setting the value of a variable whose data type is a reference type to _____ *

- ☒ 1. null
- ☐ 2. NAN
- ☐ 3. 0
- ☐ 4. None of these

Breakdown of Each Option:

null –

Setting a reference variable to null removes its association with an object.

Example:

java

```
MyClass obj = new MyClass(); // obj refers to an object
```

```
obj = null; // obj no longer refers to any object
```

NAN – Incorrect

NaN (Not a Number) is used in floating-point arithmetic, not for object references.

0 – Incorrect

0 is a numeric value, not a valid way to drop an object reference.

Explanation:

A Marker Interface in Java is an interface that does not contain any methods or fields but serves as a signal to the JVM or other code that a class implementing it has a specific capability.

✓ Which of the following is not a Marker Interface? *

- ☐ 1. Serializable
- ☐ 2. Clonable
- ☐ 3. Remote
- ☒ 4. Externlizable

Breakdown of Each Option:

Serializable – Marker Interface

Used to indicate that a class can be serialized (converted into a byte stream).

Defined in java.io.Serializable.

Cloneable – Marker Interface

Used to indicate that a class supports cloning via the clone() method.

Defined in java.lang.Cloneable.

Remote – Marker Interface

Used in Java RMI (Remote Method Invocation) to indicate that an object can be accessed remotely.

Defined in java.rmi.Remote.

Externalizable – Not a Marker Interface

Unlike marker interfaces, Externalizable contains methods (writeExternal() and readExternal()).

It requires explicit implementation of serialization logic, making it not a true marker interface.

Explanation:
In Java, keywords are reserved words that have a predefined meaning in the language and cannot be used as identifiers (such as variable names, method names, or class names).

✓ Which is a keyword? * 1/1

☐ 1. string

☐ 2. unsigned

☐ 3. Float

☒ 4. this

Breakdown of Each Option:

string – Not a keyword
Java uses String (capitalized) as a class, but string (lowercase) is not a reserved keyword.

unsigned – Not a keyword
Java does not support unsigned data types, so unsigned is not a keyword.

Float – Not a keyword
Java has a primitive type float, but Float (capitalized) is a wrapper class, not a keyword.

this – Keyword
this is a reserved keyword in Java, used to refer to the current instance of a class.

Explanation:
In Java, interface methods are implicitly public and abstract.

When a class implements an interface, it must provide a concrete implementation for all interface methods.

Since interface methods are always public, the implementing class must declare them as public, or else the compiler will throw an error.

✓ When we implement an interface method, it should be declared as public. * 1/1

☒ 1.True

☐ 2.False

☐ 3.I dont like public

☐ 4.I like my privacy

Breakdown of Each Option:

True – Correct

Interface methods must be declared as public when implemented.

False – Incorrect

If an interface method is implemented without public, it results in a compilation error.

✓ Set is the name of collection Interface used to maintain unique element. * 1/1

- ☒ 1.True ✓
- ☐ 2.False
- ☐ 3.Collection ke mummy papa se puncho
- ☐ 4.I just know "I am unique"

Explanation:

The Set interface in Java is part of the java.util package and extends the Collection interface.

It is specifically designed to store unique elements, meaning duplicates are not allowed.

Common implementations of Set include:

HashSet – Unordered collection of unique elements.

LinkedHashSet – Maintains insertion order.

TreeSet – Maintains sorted order.

Breakdown of Each Option:

True – Correct

Set ensures uniqueness of elements.

False – Incorrect

Set does not allow duplicate elements, making this option incorrect

In Java, method overriding requires that the return type in the subclass method must be compatible with the return type of the superclass method.

The superclass method test() returns int, while the subclass method test() returns short.

Since short is not a covariant return type of int, this causes a compilation error.



*

What is the result of this program?

```
class Over
{
    public static void main(String[] args){
        Under u = new Under();
        u.test();
    }
    int test(){
        System.out.println("over");
        return 1;
    }
}

class Under extends Over{
    short test(){
        super.test();
        System.out.println("Under");
        return 1;
    }
}
```

- ☐ 1. This code compiles, runs and displays over followed by Under
- ☐ 2. This code compiles, runs and displays Under followed by over
- ☒ 3. This code does not compile
- ☐ 4. Code will compile but gives runtime error



Breakdown of Each Option:

This code compiles, runs and displays "over" followed by "Under" – Incorrect

The code does not compile due to the return type mismatch.

This code compiles, runs and displays "Under" followed by "over" – Incorrect

The code does not compile, so no output is produced.

This code does not compile – Correct

The return type mismatch in method overriding causes a compilation error.

Code will compile but gives runtime error – Incorrect

The error occurs at compile time, not runtime.

Explanation:
In Java, a valid String declaration must follow proper syntax rules.

String s1 = null; is correct because:

null is a valid value for a reference type.

It means s1 does not reference any object.

Explanation:
In an online multiplayer game, different types of events—such as player movements, enemy actions, and projectile collisions—must be processed in real-time.

Priority scheduling ensures that critical events (e.g., collision detection, player inputs) are processed before less urgent tasks (e.g., background updates or animations).

This helps maintain smooth gameplay and low latency, which is crucial for multiplayer interactions.

✓ Which is valid declaration of a String? *

- ☐ 1. String s2 = 'null';
- ☐ 2. String s3 = (String) 'abc';
- ☒ 3. String s1 = null;
- ☐ 4. String s4 = (String) '\ufeed';

Breakdown of Each Option:

String s2 = 'null'; – Incorrect
Single quotes (') are used for char literals, not String literals. Strings must be enclosed in double quotes (").

String s3 = (String) 'abc'; – Incorrect
'abc' is not a valid char literal. Casting a char to String is not allowed.

String s1 = null; – Correct
null is a valid reference value for a String.

String s4 = (String) '\ufeed'; – Incorrect
Casting a Unicode character (\ufeed) to String is invalid. Unicode characters must be enclosed in double quotes.

✓ In an online multiplayer game, different types of game events, such as player movements, enemy actions, and projectile collisions, need to be processed. Which scheduling algorithm would be most suitable to ensure timely processing of critical game events? *1/1

- ☐ 1. First-Come, First-Served (FCFS) scheduling
- ☐ 2. Round Robin (RR) scheduling
- ☒ 3. Priority scheduling
- ☐ 4. Shortest Job Next (SJN) scheduling

Breakdown of Each Option:

First-Come, First-Served (FCFS) scheduling – Not suitable
Processes events in the order they arrive, which can cause delays for high-priority tasks.

Round Robin (RR) scheduling – Not optimal

Assigns equal time slices to all tasks, which may not prioritize critical events efficiently.

Priority scheduling – Correct

Assigns higher priority to critical game events, ensuring timely execution.

Shortest Job Next (SJN) scheduling – Not ideal

Prioritizes shortest tasks first, but does not account for urgency.

Explanation:
Let's break down the execution step by step:

```
java
String x = "xyz";
x.toUpperCase(); // Converts "xyz"
to "XYZ", but the result is not stored
in x
String y = x.replace('Y', 'y'); // No 'Y'
exists in "xyz", so y remains "xyz"
y = y + "abc"; // Concatenates "abc"
to "xyz", resulting in "xyzabc"
System.out.println(y); // Prints "xyzabc"
```

What is the result? *

String x = "xyz";
x.toUpperCase();
String y = x.replace('Y','y');
y = y + "abc" ;
System.out.println(y);

☐ 1. abcXYZ

☐ 2. abcxyz

☒ 3. xyzabc

☐ 4. compilation fails

Breakdown of Each Option:
abcXYZ – Incorrect 1/1

The string "xyz" was never converted to "XYZ".

abcxyz – Incorrect

"abc" is appended at the end, not at the beginning.

xyzabc – Correct

"xyz" remains unchanged, and "abc" is appended.

Compilation fails – Incorrect ✓

The code compiles successfully.

Key Observations:
x.toUpperCase() does not modify x because String objects are immutable in Java. The method returns a new string "XYZ", but since the result is not assigned to x, x remains "xyz".

x.replace('Y', 'y') does nothing because "xyz" does not contain 'Y'. So, y remains "xyz".

y = y + "abc" performs string concatenation, resulting in "xyzabc".

Final output: "xyzabc".

Explanation:
The fork() system call in Linux/Unix is used to create a new child process.

When fork() is called, it returns different values depending on whether the process is the parent or the child

Return Values of fork():
Parent Process:

fork() returns the Process ID (PID) of the child to the parent.

This allows the parent to track and manage the child process.

Child Process:

fork() returns 0 to the child process.

This helps the child process identify itself and execute its own instructions.

Failure Case:

If fork() fails to create a child process, it returns -1.

✗ What is the return value of the fork() system call in the child process? *

- ☐ 1. 0
- ☐ 2. -1
- ☐ 3. Process ID (PID) of the child
- ☒ 4. Process ID (PID) of the parent

Correct answer

- ☒ 1. 0

Breakdown of Each Option:

0 – Correct

The child process receives 0 as the return value.

-1 – Incorrect

-1 is returned only if fork() fails.

Process ID (PID) of the child – Incorrect

The parent receives the child's PID, not the child itself.

Process ID (PID) of the parent – Incorrect

The child process can retrieve the parent's PID using getpid(), but fork() does not return it.

Feedback

Difficulty level of mock *

Tough ▼

How was your experience ?

should revise some concepts from both the modules

From now onwards, I will give my best in everything in my life without any excuses. Because I know, problem is the part and parcel of life. We should always look for solutions. *

☒ I PROMISE

This content is neither created nor endorsed by Google. - [Terms of Service](#) - [Privacy Policy](#).

Does this form look suspicious? [Report](#)

Google Forms

***Explain the following text
primarily based on the
surrounding page content:***