AP Endsem Rubric		
Question	**Components**	**Marks**

1	a) The execution time is equal to **15 seconds** **(1, binary marking)**
	
	b) updated code is as follows:
	
	class APEndSemExam implements Runnable{
	public int result;
	public APEndSemExam(int res){this.result = res;}
	public void run(){. **(0.25, binary marking)**
	int a, b;
	if(Thread.getName().equals("T2")) a = towerOfHanoiSteps(10); // 10 sec. **(0.25, binary marking)**
	else b =towerOfHanoiSteps(8); // 5 sec **(0.25, binary marking)**
	… }
	synchronized(this){
	this.result = a + b **(0.25, binary marking)**
	}
	}
	
	NOTE:- changes in blue should be present .
	
	c) Execution time of updated code: **10 seconds** **(1 only if answer is correct and part b is correct).**
		3

2	- output of statement 2 is True **(0.5, binary marking)** - output of statement 3 is False **(0.5, binary marking)** - output of statement 4 is True**(0.5, binary marking)** - **1** kind of exception i.e. **NullpointerException** which occurs **3** times . **(0.5+0.5+0.5, binary)**	3
3	- creating a LinkedList using Java collection Framework (1). - Adding the 3 pairs (5,7), (2,-6), (-4,-5) to the created LinkedList (1). - Iterating over the linked list to compute w1e1+w2e2+w3e3 (1).	3
4	- 4.1 definition of static member (single copy for all objects of class) (0.5) - 4.1 practical use case (eg: int totalStudents for Student class). Give marks for any reasonable example written either in words or in code (1) - 4.2 definition of static method (can call method without creating object of the class) (0.5) - 4.2 main method is static because it serves as the entry point for a java program and the compiler should be able to execute it without creating objects of any class (1)	3

5	- encapsulation primarily focus on data hiding and presesnting the final product as a single entity to the user whereas abstraction deals with "what is being done" instead of "how it is being done" thus providing services as a blackbox to the user. - 5.1 abstraction because we are concerned with the "what" (delicious coffee should be made) and not the "how" (not interested in what the machine does inside) (1) - 5.2 polymorphism as one Peter plays many roles (1) - 5.3 encapsulation as patient receives the final pill and details of all the drugs inside it are hidden (1) - note: answers to parts 5.1 and 5.3 can be swapped by some students. In that case, give marks only if they have given proper justifications. Polymorphism is the only acceptable answer for 5.2 - note: for each of the 3 parts, give 0 marks if justification is incorrect. Also note that all three parts should have different answers (No OOPS concept should be repeated)	3
6	- No, Java class cannot extend multiple classes. One reason can be: suppose a child class extends two parent classes which have the exact same method (return type and parameters) but different implementations and child class does not override it, then if an object of child class calls that method, there will inconsistency as the method can belong to either of the parent classes. This is called the Diamond Problem in Java. Give marks for any other suitable reason or example. No marks for reasons like: efficiency, memory management etc. (1) - Yes, Java class can implement multiple interfaces because interfaces serve as a contract which enforce the implementation of a function due to which the aforementioned issue of inconcistency does not arise here. Give marks for any other valid reason. (1) - No, there will be no error because the class EndSemPreparation is implementing methods enforced by both the interfaces (which happen to be exactly the same). Since the child class has overriden the parent interfaces' method, there will not be any incosistency here, thus no chance of a Diamond Problem. (1) - For each of the three parts, give 0 marks if justification is incorrect	3

7	- blank-1 => Comparator<Book> (binary marking, case sensitive) (0.5)
	
	- blank-2 => compare (binary marking, case sensitive) (0.5)
	
	- blank-3 => can be written in many ways, 1 example given below (2 = 1 mark for correct sorting for price + 1 mark for correct sorting for type)
	
	- code:
	
	if(b1.price != b2.price){return -1;}
	if(b1.type.equals("hardcover")){return -1;}
	if(b1.type.equals("paperback") && b2.type.equals("ebook")){return -1;}
	return 0; // this can be anything based on assumption
	
	- note: some students might have assumed opposite priority wrt book type. If the student has clearly mentioned their assumption and written the correct code according to that then give full marks. If no assumption is written but the code for opposite priority wrt book type is written, give 0 marks.
		3

8	- 8.1 Different. instanceof also checks for subtypes whereas getClass() checks for an exact match (any other relevant example / point is acceptable) (1) - 8.2 Different. AssertEquals uses .equals() for comparison of objects whereas AssertSame uses == operator for comparison of objects. Any relevant example / point highlighting the same is acceptable (1) - 8.3 Different. Association implies that an object of class A is a data member in class B whereas Dependency implies that an object of class A is passed as a parameter in a method in class B (any other difference acceptable, there can be many answers to this) (1) - for each of the three parts, if reason is not correct / example is not reasonable, give 0 marks	3
9	- wildcards, denoted by '?' are an aspect of generic programming which help put constraints such as upper and lower bound wrt classes on a variable which is not possible with simple generic programming. Any other valid reason is acceptable. (1) - 9.1 classes A, C, D [follow binary marking] (1) - 9.2 classes A, B, C, D [follow binary marking] (1)	3
10	- Defining abstract class Animal/ public abstract class Animal (1). - Declaring public abstract void animalSound(); (1) - Defining public void sleep(){System.out.println("Zzz");} (1)	3

11	- Point p = new Point(); a = p.calcDist(3,5,-3,-3); b = p.calcDist(-1,2,11,7); The code can vary, but if calcDist() is called with these 2 set of arguments, award 1 mark. (1) - assertEquals(10, a); 1 mark for this check. (1) - assertEquals(13,b); 1 mark for this check. (1)	3
12	**"YES, the program will run." (1, binary)** **Output to the program is :-** Welcome to the Exam4 **(0.5, binary)** true **(0.5, binary)** false **(0.5, binary)** This is function 3! **(0.5, binary)**	3
13	**Output to the program is :-** 3 **(0.5, binary)** It's Object method **(0.5, binary)** It's String method **(0.5, binary)** It's Object method **(0.5, binary)** **NOTE:- order of the output does matter, so make sure order is correct.** The concepts used in the above program is: a) Functional Overriding/runtime polymorphism **(1, binary).** b) Functional Overloading/compile time polymorphism **(1, binary)** **Explanation**(**not to be graded**): The **functional overloading** is shown using the function **'f'**, which has the function declaration but the difference is the parameters passed. **Functional Overriding** is shown using the function '**g**'.	4

14	**There are a total of 4 errors:-** 1) First 2 Errors, are the **incorrect use of modifier i.e. private (0.5, binary) and static (0.25, binary) in abstract class** vehicle class method **"changeGear"** and **missing abstract keyword (0.25, binary)** in its declaration. 2) Third Error, is the **incorrect use of modifier "private"** in **"move" method declaration** inside the **interface Transport**. **(0.5, binary)** 3) Fourth Error, is the **incorrect declared type in the main function i.e. "Transport" (0.5, binary)** and it's trying to call **"changeBreak"** method. **Fix for the errors:-** 1) Replace the keywords "**private static**" in the **abstract class** method "**changeGear**" by keyword "**public**" **(1, binary)** 2) Replace the keyword "**private**" in the **interface** method "**move**" by keyword "**public**" **(0.5, binary)** 3) Replace the declared type "**Transport**" in the main function, by either "**Vehicle**" or "**Bike**" **(0.5, binary)** **NOTE:- if corrections are shown via code snippets, make sure the above mentioned corrections are present in the code.**	4
15	1. **Encapsulation** to be followed: **All attributes** to be **private (1, binary)** . 2. **Final**: **All class attributes** should be marked as **final (1, binary)** 3. **No setters** should be present in the code. **(1, binary)** 4. **Class should be made final (1, binary)**	4
16	**Rubric for the UML diagram**:- link	4
17	- BufferedInputStream input = new BufferedInputStream(file); (1) - int i = input.read(); while(i!=-1 or !EOF) (0.5 for this condition check) - The body of while loop should have 2 things: System.out.print((char) i) to print the read byte, and i = input.read() to read the next byte in the file. Award 1 mark for each of these things. (2) - input.close() to close the stream Reader (0.5).	4

18	- SleepExp1 thread1 = new SleepExp1(); Thread is in NEW STATE (1). - thread1.start(); Thread moves to RUNNABLE state. The scheduler picks it up when it is free, and then the thread would be in RUNNING state (1). - The loop is executed 4 times. In each iteration, Thread.sleep(500) would send the thread to BLOCKED/NON-RUNNABLE state for 0.5 seconds. (1) - After 0.5 seconds, thread moves back to RUNNABLE state, and when the scheduler picks it up, it goes to RUNNING state. After 4 iterations, thread reaches the TERMINATED/DEAD state (1).	4
19	- Defining the custom InvalidTypeException class extending RuntimeException (1.5) - If object **instanceof** Integer/ object.getClass==Integer.class {throw new InvalidTypeException("Message");} (1.5) - The class has one generic attribute (0.5) - this.field = object assignment done if object is NOT of Integer type (0.5) (Points 2 and 4 should be shown in the parametrized constructor of Point class, taking object as argument (the name of the argument passed can be different))	4
20	**There are 5 design patterns used -** **Singleton, Factory, Template, Iterator, and Flyweight. (1*5 =5 marks, 1 for mentioning each design pattern)** 1. **Singleton** : It is seen in **class Exam2** where only **one instance of type Exam2** is created and its **constructor has also been made private**. The **static** method **Exam_2** has been used and it creates a new instance only if the **static field exam2** is null otherwise it returns the same instance. **(1 mark for explanation)** 2. **Template** : **Class Exam1** is **abstract** and it has **two** classes inheriting from it and **overrides** one function (**abstract method**) but contains the same definition of the non abstract method.**(1 mark for explanation)** 3. **Factory** : In class **Exam4**, the method **method_exam4** is responsible for **creation of objects of inherited classes** of type **Exam1 (1 mark for explanation)** 4. **Iterator** : iterating through **collections** and the **type of iterator is generic**. **(1 mark for explanation)** 5. **Flyweight** : The function **function3() is flyweighted due to** implementation of **string in JVM**. **(1 mark for explanation)**	10