CMR INSTITUTE OF TECHNOLOGY
Department of MCA
Odd Semester 2018
Internal Examination – II (Key)

Semester : III A
Subject Code : 17MCA32
Subject Name : Java Programming

Q1(a) Define a package. Explain the creation of package and create the package by the name shape to illustrate it.
Ans: A java package is a group of similar types of classes, interfaces and sub-packages.
Advantage of Java Package
1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.
Creation of Package:
While creating a package, we should choose a name for the package and include a package statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that we want to include in the package.
The package statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.
If a package statement is not used then the class, interfaces, enumerations, and annotation types will be placed in the current default package.
EX:

```
package shape;
public class circle {
   public void display1()
   {
       System.out.println("formula of circle is 3.142*r*r");
   }
}
package shape;
public class triangle {
   public void display3()
   {
      System.out.println("formula of triangle =0.5*length*breadth");
   }
}
package shape;
public class square {
   public void display2()
   {
       System.out.println("formula of Square =lengthv*width")
   }
 }
import shape.*;
public class prgrm7
{
   public static void main(String arg[])
   {
      circle ob1 = new circle();
      square ob2 =new square();
      triangle ob3 =new triangle();
      ob1.display1();
      ob2.display2();
      ob3.display3();
```

```
    }
}
```

Q1(b) Explain access specifiers in java with an example.
Ans:
The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.
There are 4 types of java access modifiers:
private
default
protected
public

Access Modifier	within class	within package	outside package by subclass only	outside pac
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

EX:

```
class A{
private A(){}//private constructor
void msg()
{
System.out.println("Hello java");
}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```

Q2(a) Explain wait(),notify(), notifyAll() in thread communication with example.

Ans: Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.
Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of Object class:
wait()
notify()
notifyAll()

---

1) wait() method
Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Description

public final void wait()throws InterruptedException	waits until object is notified.
public final void wait(long timeout)throws InterruptedException	waits for the specified amount of time.

2) notify() method
Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:
public final void notify()

3) notifyAll() method
Wakes up all threads that are waiting on this object's monitor. Syntax:
public final void notifyAll()

Q2(b) What is thread priority? Explain it with an example.

Ans: Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defiend in Thread class:

public static int MIN_PRIORITY
public static int NORM_PRIORITY
public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY).
The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example of priority of a Thread:

```
class TestMultiPriority1 extends Thread{
 public void run(){
   System.out.println("running thread name is:"+Thread.currentThread().getName());
   System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
  }
 public static void main(String args[]){
  TestMultiPriority1 m1=new TestMultiPriority1();
  TestMultiPriority1 m2=new TestMultiPriority1();
 m1.setPriority(Thread.MIN_PRIORITY);
  m2.setPriority(Thread.MAX_PRIORITY);
  m1.start();
  m2.start();
 }
}
```

Q3(a) Write the differences between abstract class and interface.

oops interface vs abstract class

Interface	Abstract class
Interface support multiple inheritance	Abstract class does not support multiple inheritance
Interface does'n Contains Data Member	Abstract class contains Data Member
Interface does'n contains Cunstructors	Abstract class contains Cunstructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Member of interface can not be Static	Only Complete Member of abstract class can be Static

Q3(b) Explain the usage of keywords try, throw, catch, finally, throws with their syntax.
Ans: Try:
Try is used to guard a block of code in which exception may occur. This block of code is called guarded region
Syn:
try
{
Stmt-1;
}
Catch:
A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in guarded code, the catch block that follows the try is checked, if the type of exception that occured is listed in the catch block then the exception is handed over to the catch block which then handles it.
Syn:
Catch(ExceptionType obj)
{
Stmt
}
Throw:
Throw keyword is used to throw an exception explicitly. Only object of Throwable class or its sub classes can be thrown. Program execution stops on encountering throw statement, and the closest catch statement is checked for matching type of exception.
Syn:
Throw new throwableinstance;
Throws:
If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature.
Syn:
type method_name(parameter_list) throws *exception_list*
{
//definition of method
}
Finally:

A finally keyword is used to create a block of code that follows a try block. A finally block of code always executes whether or not exception has occurred. Using a finally block, lets you run any cleanup type statements that you want to execute, no matter what happens in the protected code. A finally block appears at the end of catch block.
Syn:
Finally
{
Block of statements
}

Q4(a) Explain enumeration using an example. Explain the usage of values() and valueOf().

Ans: **Enum in java** is a data type that contains fixed set of constants.
Java Enums can be thought of as classes that have fixed set of constants.
enum improves type safety
enum can be easily used in switch
enum can be traversed
enum can have fields, constructors and methods
enum may implement many interfaces but cannot extend any class because it internally extends Enum class

```
class EnumExample1{
public enum Season { WINTER, SPRING, SUMMER, FALL }

public static void main(String[] args) {
for (Season s : Season.values())
System.out.println(s);

}}
```

Q4(b) Explain Auto boxing and Unboxing with an example.

Ans: Autoboxing and Unboxing:
The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing. This is the new feature of Java5. So java programmer doesn't need to write the conversion code.
Advantage of Autoboxing and Unboxing:
No need of conversion between primitives and Wrappers manually so less coding is required.

Simple Example of Autoboxing in java:

```
class BoxingExample1{
  public static void main(String args[]){
    int a=50;
        Integer a2=new Integer(a);//Boxing

        Integer a3=5;//Boxing

        System.out.println(a2+" "+a3);
 }
}

  class UnboxingExample1{
  public static void main(String args[]){
    Integer i=new Integer(50);
        int a=i;
```

```
        System.out.println(a);
 }
}
```

5(a)How is multiple inheritance achieved in java? Explain with an example program

Multiple inheritance in java is achieved through interfaces

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}
public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

(b) What is an exception? Explain multiple catch block statements with an example program

A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created and *thrown* in the method that caused the error. That method may choose to handle the exception itself, or pass it on.

In some cases, more than one exception could be raised by a single piece of code. To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception. When an exception is thrown, each catch statement is inspected in orderand the first one whose type matches that of the exception is executed. After one catch statement executes, the others are bypassed, and execution continues after the try/catch block. The following example traps two different exception types:

```
// Demonstrate multiple catch statements.
class MultiCatch {
public static void main(String args[]) {
try {
}
}
int a = args.length;
System.out.println("a = " + a);
int b = 42 / a;
int c[] = { 1 };
c[42] = 99;
} catch(ArithmeticException e) {
System.out.println("Divide by 0: " + e);
} catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Array index oob: " + e);
}
System.out.println("After try/catch blocks.");
}
}
```

Q6(a) Explain the way of creating our own exception sub class with an example of queue which is throwing queue full and queue is empty exception.

Ans: We can also create our own exception sub class by extending the Exception class. We can define a constructor for our exception subclass and we can override toString() to display our customized message

```
class ExcQueue extends Exception
{
   ExcQueue(String s)
   {
      super(s);

   }
}
class Queue
{
   int front,rear;
   int q[]=new int[10];
   Queue()
   {
      rear=-1;
      front=-1;
   }
   void enqueue(int n) throws ExcQueue
   {
      if(rear==9)
         throw new ExcQueue("queue is full");
      rear++;
      q[rear]=n;
      if(front==-1)
         front=0;
   }
   int dequeue() throws ExcQueue
   {
      if(front==-1)
         throw new ExcQueue("queue is empty");
      int temp=q[front];
      if(front==rear)
         front=rear=-1;
      else
         front++;
      return(temp);
   }
}
public class prog6 {
public static void main(String arg[])
{
   Queue a=new Queue();
   try
   {
      a.enqueue(6);
      a.enqueue(27);
   }
   catch(ExcQueue e)
   {
      System.out.println(e.getMessage());

   }
   try
   {

      System.out.println(a.dequeue());
```

```
        System.out.println(a.dequeue());
        System.out.println(a.dequeue());


    }
    catch(ExcQueue e)
    {
        System.out.println(e.getMessage());
    }
}
}
```

Q7(a) List some of the most common types of exceptions that might occur in java with example.

Ans

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

ArithmeticException

It is thrown when an exceptional condition has occurred in an arithmetic operation.

ArrayIndexOutOfBoundException

It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

ClassNotFoundException

This Exception is raised when we try to access a class whose definition is not found

FileNotFoundException

This Exception is raised when a file is not accessible or does not open.

IOException

It is thrown when an input-output operation failed or interrupted

InterruptedException

It is thrown when a thread is waiting , sleeping , or doing some processing , and it is interrupted.

NoSuchFieldException

It is thrown when a class does not contain the field (or variable) specified

NoSuchMethodException

It is thrown when accessing a method which is not found.

NullPointerException

This exception is raised when referring to the members of a null object. Null represents nothing

NumberFormatException

This exception is raised when a method could not convert a string into a numeric format.

RuntimeException

This represents any exception which occurs during runtime.

StringIndexOutOfBoundsException

It is thrown by String class methods to indicate that an index is either negative than the size of the string

Examples of Built-in Exception:

Arithmetic exception

```
// Java program to demonstrate ArithmeticException
class ArithmeticException_Demo
{
    public static void main(String args[])
    {
        try {
            int a = 30, b = 0;
            int c = a/b;  // cannot divide by zero
            System.out.println ("Result = " + c);
        }
        catch(ArithmeticException e) {
            System.out.println ("Can't divide a number by 0");
        }
    }
```

```
        }
```
ArrayIndexOutOfBoundException

```
class ArrayIndexOutOfBound_Demo
{
   public static void main(String args[])
   {
      try{
         int a[] = new int[5];
         a[6] = 9; // accessing 7th element in an array of
                  // size 5
      }
      catch(ArrayIndexOutOfBoundsException e){
         System.out.println ("Array Index is Out Of Bounds");
      }
  }
```

7(b) What is an interface? How do you create and implement interfaces in java?

An interface in java is a blueprint of a class. It has static constants and abstract methods only.

The interface in java is a mechanism to achieve fully abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.

Java Interface also represents IS-A relationship.

It cannot be instantiated just like abstract class.

There are mainly three reasons to use interface. They are given below.

It is used to achieve fully abstraction.

By interface, we can support the functionality of multiple inheritance.

It can be used to achieve loose coupling.

Defining an Interface

An interface is defined much like a class. This is the general form of an interface:

```
access interface name {
return-type method-name1(parameter-list);
return-type method-name2(parameter-list);
type final-varname1 = value;
type final-varname2 = value;
// ...
return-type method-nameN(parameter-list);
type final-varnameN = value;
}
```

Implementing Interfaces

Once an interface has been defined, one or more classes can implement that interface. To implement an interface, include the implements clause in a class definition, and then create the methods defined by the interface.

The general form of a class that includes the implementsclause looks like this:

```
class classname [extends superclass] [implements interface [,interface...]] {
// class-body
}
```

EX:

```
interface Printable
{
void print();
}

interface Showable
{
void show();
}
class A7 implements Printable,Showable
```

```
{
  public void print()
{
System.out.println("Hello");
          }
public void show()
{
System.out.println("Welcome");
}

public static void main(String args[])
{
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

Q8(a) Implement the producer-consumer problem with a program.

Ans:

```
class Q {
int n;
boolean valueSet = false;
synchronized int get() {
if(!valueSet)
try {
wait();
} catch(InterruptedException e) {
System.out.println("InterruptedException caught");
}
System.out.println("Consumed: " + n);
valueSet = false;
notify();
return n;
}
synchronized void put(int n) {
if(valueSet)
try {
wait();
} catch(InterruptedException e) {
System.out.println("InterruptedException caught");
}
this.n = n;
valueSet = true;
System.out.println("Producer: " + n);
notify();
}
}
class Producer implements Runnable {
Q q;
Producer(Q q) {
this.q = q;
new Thread(this, "Producer").start();
}
public void run() {
int i = 0;
while(true) {
```

```
q.put(i++);
}
}
}
class Consumer implements Runnable {
Q q;
Consumer(Q q) {
this.q = q;
new Thread(this, "Consumer").start();
}
public void run() {
while(true) {
q.get();
}
}
}
class PCFixed {
public static void main(String args[]) {
Q q = new Q();
new Producer(q);
new Consumer(q);
System.out.println("Press Control-C to stop.");
}
}
```

Q9(a) Explain the two ways of creating threads with an example

Ans: There are two ways to create a thread:

By extending Thread class

By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

Thread()
Thread(String name)
Thread(Runnable r)
Thread(Runnable r,String name)

1.By extending Thread class:

The steps for creating a thread by using Thread class

1.Create a class that extends thread class.

2. Override run()

3.Create an object for that class

4.Call start()

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

2.By implementing Runnable interface:

The steps for creating a thread by using runnable interface

1.Create a class that implements runnable interface

2. Override run()

3.Create an object for the user defined sub class
4.Create an object for thread class and pass user defined class object to its constructor
5.Call start()
EX:

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
}
}
```

Q9(b) Explain how java achieves synchronization using an example.
Ans: Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.
Java Synchronization is better option where we want to allow only one thread to access the shared resource.
The synchronization is mainly used to
To prevent thread interference.
To prevent consistency problem.
Java synchronized method
If you declare any method as synchronized, it is known as synchronized method.
Synchronized method is used to lock an object for any shared resource.
When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
 Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
```

```
public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Synchronized block in java

Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

```
synchronized (object reference expression) {
  //code block
}
```

Ex:

```
class Table{

 void printTable(int n){
   synchronized(this){//synchronized block
     for(int i=1;i<=5;i++){
      System.out.println(n*i);
      try{
       Thread.sleep(400);
      }catch(Exception e){System.out.println(e);}
     }
   }
 }//end of the method
}
  class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
 public class TestSynchronizedBlock1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
```

```
t2.start();
}
```
Q10 WRITE A JAVA PROGRAM WHICH HAS
A CLASS CALLED ACCOUNT THAT CREATES ACCOUNT WITH 500RS MINIMUM BALANCE, A DEPOSIT() METHOD TO DEPOSIT AMOUNT, A WITHDRAW() METHOD TO WITHDRAW AMOUNT AND ALSO THROWS LESSBALANCEEXCEPTION IF AN ACCOUNT HOLDER TRIES TO WITHDRAW MONEY WHICH MAKES THE BALANCE BECOME LESS THAN 500RS.
A CLASS CALLED LESSBALANCEEXCEPTION WHICH RETURNS THE STATEMENT THAT SAYS WITHDRAW AMOUNT (___RS) IS NOT VALID.A CLASS WHICH CREATES 2 ACCOUNTS, BOTH ACCOUNT DEPOSIT MONEY AND ONE ACCOUNT TRIES TO WITHDRAW MORE MONEY WHICH GENERATES A LESSBALANCEEXCEPTION TAKE APPROPRIATE ACTION FOR THE SAME.

SOURCE CODE:

```
import java.io.*;
class LessBalanceException extends Exception
{
LessBalanceException(double amt)
{
System.out.println("Withdrawing"+amt+"is invalid");
}
}
class Account
{
static int count=124; int an;
double bal; String name;
Account(double bal,String n)
{
System.out.println("\nNew Account opened...!!!\n");
this.bal=bal;
count++;
an=count;
name=n;
}
void deposit(double amt)
{
System.out.println("Availabe Balance :"+bal); bal=bal+amt;
System.out.println("Rs." +amt +"/- Credited"); System.out.println("Balance :"+bal);
}
void withdraw(double amt) throws LessBalanceException
{
System.out.println("\nAvailable Balance :"+bal); bal=bal-amt;
if(bal<500)
{
bal=bal+amt;
throw new LessBalanceException(amt);
}
System.out.println("Rs."+amt+"/-Debited"); System.out.println("Balanc :\n"+bal);
}
void Balance()
{
System.out.println("\n**********Customer
information**********\n");
System.out.println("Customer Name  : "+name);
System.out.println("Customer Id     : "+an);
System.out.println("Balance           : "+bal);
}
}
```

```
class AccountDemo1
{
static int i=0;
public static void main(String arg[])throws IOException
{
Account ob[] = new Account[10]; BufferedReader br= new BufferedReader(newInputStreaRader(System.in));
double amt; String name;
boolean b= true,f=false; int k;
while(b)
{
System.out.println("\n***Bank Transaction***\n"); System.out.println("1.Open new
Account\n2.Deposit\n3.Withdraw\n4.Balance\nExit"); System.out.println("Enter Your Choice\n");
ch=Integer.parseInt(br.readLine()); switch(ch)
{
case 1:
System.out.println("Opening New Account\n"); System.out.println("Enter your name :\n"); name=br.readLine();
System.out.println("Enter initial amount(> Rs.500/-)"); amt=Double.parseDouble(br.readLine());
ob[i]=new Account(amt,name); i++; break;
case 2:
System.out.println("Enter Account number\n"); an=Integer.parseInt(br.readLine()); for(k=0;k<2;k++)
if(an==ob[k].an)
{
f=true; break;
}
if(f)
{
System.out.println("Enter the Amount for Deposit :"); amt=Double.parseDouble(br.readLine()); ob[k].deposit(amt);
}
else
System.out.println("Invalid Account Number....!!!");
break; case 3:
System.out.println("Enter Account number\n"); an=Integer.parseInt(br.readLine()); for(k=0;k<2;k++)
if(an==ob[k].an)
{
f=true; break;
}
if(f)
{
System.out.println("Enter the Amount for Withdraw :"); amt=Double.parseDouble(br.readLine());
try
{
ob[k].withdraw(amt);
}
catch(LessBalanceException e)
{
}
}
Else
System.out.println("Invalid Account Number....!!!");

break; case 4:

System.out.println("Enter Account number\n"); an=Integer.parseInt(br.readLine()); for(k=0;k<2;k++)
if(an==ob[k].an)
{
```

```
f=true; break;
}

if(f)
{

ob[k].Balance();
}
else
System.out.println("Invalid Account Number....!!!");

break; case 5:
b=false;

System.exit(1);
default: System.out.println("Invalid Choice !!!\n");
}
}

}
}
```