**1Wrapper classes**

1. **Check if character is a Digit**
2. Compare two Strings
3. Convert using valueof method
4. Create Boolean Wrapper usage
5. Convert null to wrapper classes

**.package** day7\_assign;

**public** **class** Assign {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

/\*char c = '5';

if (Character.isDigit(c)) {

System.out.println(c+"is a digit.");

} else {

System.out.println(c+"is not a digit.");

}\*/

/\*String str1="Hello";

String str2="Hello";

String str3="World";

System.out.println(str1.equals(str2));

System.out.println(str1.equals(str3));

System.out.println(str1.compareTo(str2));

System.out.println(str1.compareTo(str3));

System.out.println(str3.compareTo(str1));

\*/

/\*

Integer intObj = Integer.valueOf("123");

System.out.println(intObj);

Double doubleObj = Double.valueOf("123.45");

System.out.println(doubleObj);

Boolean boolObj = Boolean.valueOf("true");

System.out.println(boolObj); // true \*/

/\*

Boolean boolObj1 = Boolean.TRUE;

Boolean boolObj2 = Boolean.valueOf(true);

Boolean boolObj3 = true;

System.out.println(boolObj1.equals(boolObj2));

System.out.println(boolObj1 == boolObj2);

boolean primitiveBool = boolObj1;

System.out.println(primitiveBool); \*/

/\*

Integer intObj = null;

Double doubleObj = null;

Boolean boolObj = null;

try {

int primitiveInt = intObj;

double primitiveDouble = doubleObj;

boolean primitiveBool = boolObj;

} catch (NullPointerException e) {

System.out.println("NullPointerException caught");

}

\*/

/\*

int x = 10;

System.out.println("Before method call: x="+x);

changeValue(x);

System.out.println("After method call: x="+x);

}

public static void changeValue(int x) {

x = 20;

System.out.println("Inside method: x="+x); \*/

/\*

int a = 10;

int b = 20;

System.out.println("Before swap:a="+a+",b="+b);

swap(a, b);

System.out.println("After swap:a="+a+",b="+b);

}

public static void swap(int a, int b) {

int temp = a;

a = b;

b = temp;

System.out.println("Inside swap:a="+a+",b="+b);

\*/

/\*

\* int a = 10;

int b = 20;

System.out.println("Before swap:a="+a+",b="+b);

swap(a, b);

System.out.println("After swap:a="+a+",b="+b);

}

public static void swap(int a, int b) {

int temp = a;

a = b;

b = temp;

System.out.println("Inside swap a="+a+",b="+b);

\*/

/\*

\* int a = 10;

double b = 20.5;

boolean c = true;

char d = 'A';

System.out.println("Before method call:");

System.out.println("a="+a);

System.out.println("b="+b);

System.out.println("c="+c);

System.out.println("d="+d);

changeValues(a, b, c, d);

System.out.println("After method call:");

System.out.println("a="+a);

System.out.println("b="+b);

System.out.println("c="+c);

System.out.println("d="+d); }

public static void changeValues(int a, double b, boolean c, char d) {

a = 20;

b = 30.5;

c = false;

d = 'B';

System.out.println("Inside method:");

System.out.println("a="+a);

System.out.println("b="+b);

System.out.println("c="+c);

System.out.println("d="+d);

\*/

}

}

**Call by Reference (Using Objects)**

1. **Create a class Box with a variable length. Write a method that modifies the value of length by passing the Box object. Show that the original object is modified.**

1. **package** day7\_assign;

**public** **class** Call\_by\_ref {

**int** length;

**public** Call\_by\_ref(**int** length) {

**this**.length=length;

}

**public** **void** printLength() {

System.***out***.println("Length"+length);

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Call\_by\_ref box = **new** Call\_by\_ref(10);

System.***out***.println("Before modification");

box.printLength();

*modifyBox*(box);

System.***out***.println("After modification");

box.printLength();

}

**public** **static** **void** modifyBox(Call\_by\_ref box) {

box.length = 20;

}

}

1. Write a Java program to pass an object to a method and modify its internal fields. Verify that the changes reflect outside the method.

2. **package** day7\_assign;

**class** Employee {

String name;

**int** age;

**public** Employee(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public** **void** printDetails() {

System.***out***.println("Name: " + name);

System.***out***.println("Age: " + age);

}

}

**public** **class** CBR\_2 {

**public** **static** **void** main(String[] args) {

Employee employee = **new** Employee("J", 30);

System.***out***.println("Before modification:");

employee.printDetails();

*modifyEmployee*(employee);

System.***out***.println("After modification:");

employee.printDetails();

}

**public** **static** **void** modifyEmployee(Employee employee) {

employee.name = "D";

employee.age = 31;

}

}

**Create a class Student with name and marks. Write a method to update the marks of a student. Demonstrate the changes in the original object3. package day7\_assign;**

**class** Student {

String name;

**double** marks;

**public** Student(String name, **double** marks) {

**this**.name = name;

**this**.marks = marks;

}

**public** **void** printDetails() {

System.***out***.println("Name"+name);

System.***out***.println("Marks"+marks);

}

}

**public** **class** CBR\_3 {

**public** **static** **void** main(String[] args) {

Student student = **new** Student("J", 85.0);

System.***out***.println("Before updating marks:");

student.printDetails();

*updateMarks*(student, 90.0);

System.***out***.println("After updating marks");

student.printDetails();

}

**public** **static** **void** updateMarks(Student student, **double** newMarks) {

student.marks = newMarks;

}

}

**Create a program to show that Java is strictly "call by value" even when passing objects (object references are passed by value).**

4. **package** day7\_assign;

**class** Dog {

String name;

**public** Dog(String name) {

**this**.name = name;

}

}

**public** **class** CBR\_4 {

**public** **static** **void** main(String[] args) {

Dog myDog = **new** Dog("Max");

System.***out***.println("Before method call: " + myDog.name);

*changeDog*(myDog);

System.***out***.println("After method call: " + myDog.name);

*changeDogReference*(myDog);

System.***out***.println("After changing reference: " + myDog.name);

}

**public** **static** **void** changeDog(Dog dog) {

dog.name = "Fido";

}

**public** **static** **void** changeDogReference(Dog dog) {

dog = **new** Dog("Rocky");

}

}

**Write a program where you assign a new object to a reference passed into a method. Show that the original reference does not change.**

5. **package** day7\_assign;

**class** Person {

String name;

**public** Person(String name) {

**this**.name = name;

}

}

**public** **class** CBR\_5 {

**public** **static** **void** main(String[] args) {

Person person = **new** Person("John");

System.***out***.println("Before method call: " + person.name);

*changePerson*(person);

System.***out***.println("After method call: " + person.name);

}

**public** **static** **void** changePerson(Person p) {

p = **new** Person("Jane");

System.***out***.println("Inside method: " + p.name);

}

}

**Explain the difference between passing primitive and non-primitive types to methods in Java with examples.**

6. **package** day7\_assign;

**public** **class** CBR\_6 {

**public** **static** **void** main(String[] args) {

**int** primitiveValue = 10;

System.***out***.println("Before method call"+primitiveValue);

*changePrimitiveValue*(primitiveValue);

System.***out***.println("After method call"+primitiveValue);

PPerson person = **new** PPerson("J");

System.***out***.println("Before method call"+person.name);

*changePersonName*(person);

System.***out***.println("After method call"+person.name);

System.***out***.println("Before reassigning"+person.name);

*reassignPerson*(person);

System.***out***.println("After reassigning"+person.name);

}

**public** **static** **void** changePrimitiveValue(**int** value) {

value = 20;

System.***out***.println("Inside method"+value);

}

**public** **static** **void** changePersonName(PPerson p) {

p.name = "m";

System.***out***.println("Inside method"+p.name);

}

**public** **static** **void** reassignPerson(PPerson p) {

p = **new** PPerson("J");

System.***out***.println("Inside method"+p.name);

}

}

**class** PPerson {

String name;

**public** PPerson(String name) {

**this**.name = name;

}

}

**Can you simulate call by reference in Java using a wrapper class or array? Justify with a program.**

7. **package** day7\_assign;

**public** **class** CBR\_7 {

**public** **static** **void** main(String[] args) {

Wrapper wrapper = **new** Wrapper(10);

System.***out***.println("Before method call"+wrapper.value);

*changeWrapperValue*(wrapper);

System.***out***.println("After method call"+wrapper.value);

**int**[] array = **new** **int**[] {10};

System.***out***.println("Before method call"+array[0]);

*changeArrayValue*(array);

System.***out***.println("After method call "+array[0]);

}

**public** **static** **void** changeWrapperValue(Wrapper wrapper) {

wrapper.value=20;

}

**public** **static** **void** changeArrayValue(**int**[] array) {

array[0]=20;

}

}

**class** Wrapper {

**int** value;

**public** Wrapper(**int** value) {

**this**.value = value;

}

}

**Multithreading**

**Write a program to create a thread by extending the Thread class and print numbers from 1 to 5.**

1. **package** day7\_assign;

**class** NumberThread **extends** Thread {

**public** **void** run() {

**for** (**int** i=1;i<=5;i++) {

System.***out***.println(i);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

}

**public** **class** Multithreading1 {

**public** **static** **void** main(String[] args) {

NumberThread thread = **new** NumberThread();

thread.start();

}

}

**Create a thread by implementing the Runnable interface that prints the current thread name.**

2. **package** day7\_assign;

**class** ThreadRunnable **implements** Runnable {

**public** **void** run() {

System.***out***.println("Current Thread Name: " + Thread.*currentThread*().getName());

}

}

**public** **class** MT\_2 {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

ThreadRunnable runnable = **new** ThreadRunnable();

Thread thread = **new** Thread(runnable);

thread.setName("MyThread");

thread.start();

}

**}**

**Write a program to create two threads, each printing a different message 5 times.**

3. **package** day7\_assign;

**class** MessageThread **implements** Runnable {

**private** String message;

**public** MessageThread(String message) {

**this**.message = message;

}

**public** **void** run() {

**for** (**int** i = 0; i < 5; i++) {

System.***out***.println(message);

**try** {

Thread.*sleep*(1000); // Sleep for 1 second

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

}

**public** **class** MT\_3 {

**public** **static** **void** main(String[] args) {

MessageThread thread1 = **new** MessageThread("Hello");

MessageThread thread2 = **new** MessageThread("World");

Thread t1 = **new** Thread(thread1);

Thread t2 = **new** Thread(thread2);

t1.start();

t2.start();

}

}

**Demonstrate the use of Thread.sleep() by pausing execution between numbers from 1 to 3.**

4. **package** day7\_assign;

**public** **class** MT\_4 {

**public** **static** **void** main(String[] args) {

**try** {

**for** (**int** i = 1; i <= 3; i++) {

System.***out***.println(i);

Thread.*sleep*(2000); // Pause execution for 2 seconds

}

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

System.***out***.println("Thread interrupted.");

}

}

}

**Create a thread and use Thread.yield() to pause and give chance to another thread.5. package day7\_assign;**

**class** YieldThread **extends** Thread {

String threadName;

YieldThread(String name) {

threadName = name;

}

**public** **void** run() {

**for** (**int** i = 0; i < 5; i++) {

System.***out***.println(threadName + ": " + i);

**if** (i == 2) {

System.***out***.println(threadName + " yielding to other threads...");

Thread.*yield*();

}

}

}

}

**public** **class** MT\_5 {

**public** **static** **void** main(String[] args) {

YieldThread thread1 = **new** YieldThread("Thread-1");

YieldThread thread2 = **new** YieldThread("Thread-2");

thread1.start();

thread2.start();

}

}

**Implement a program where two threads print even and odd numbers respectively.**

**6.**

**package** day7\_assign;

**class** EvenThread **extends** Thread {

**public** **void** run() {

**for** (**int** i = 2; i <= 10; i += 2) {

System.***out***.println("Even Thread: " + i);

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

}

**class** OddThread **extends** Thread {

**public** **void** run() {

**for** (**int** i = 1; i <= 9; i += 2) {

System.***out***.println("Odd Thread: " + i);

**try** {

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

}

**public** **class** MT\_6 {

**public** **static** **void** main(String[] args) {

EvenThread evenThread = **new** EvenThread();

OddThread oddThread = **new** OddThread();

evenThread.start();

oddThread.start();

}

}

7. **Create a program that starts three threads and sets different priorities for them.**

**package** day7\_assign;

**class** PriorityThread **extends** Thread {

String threadName;

PriorityThread(String name) {

threadName = name;

}

**public** **void** run() {

**for** (**int** i = 0; i < 5; i++) {

System.***out***.println(threadName + ": " + i);

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

}

**public** **class** MT\_7 {

**public** **static** **void** main(String[] args) {

PriorityThread highPriorityThread = **new** PriorityThread("High Priority Thread");

PriorityThread normalPriorityThread = **new** PriorityThread("Normal Priority Thread");

PriorityThread lowPriorityThread = **new** PriorityThread("Low Priority Thread");

highPriorityThread.setPriority(Thread.***MAX\_PRIORITY***);

normalPriorityThread.setPriority(Thread.***NORM\_PRIORITY***);

lowPriorityThread.setPriority(Thread.***MIN\_PRIORITY***);

highPriorityThread.start();

normalPriorityThread.start();

lowPriorityThread.start();

}

}

8. **Write a program to demonstrate Thread.join() – wait for a thread to finish before proceeding.**

**package** day7\_assign;

**class** JoinThread **extends** Thread {

String threadName;

JoinThread(String name) {

threadName = name;

}

**public** **void** run() {

**for** (**int** i = 0; i < 5; i++) {

System.***out***.println(threadName + ": " + i);

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

}

**public** **class** MT\_8 {

**public** **static** **void** main(String[] args) {

JoinThread thread = **new** JoinThread("Child Thread");

thread.start();

**try** {

thread.join();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

System.***out***.println("Main thread finished waiting for child thread.");

System.***out***.println("Main thread exiting.");

}

}

**9 Show how to stop a thread using a boolean flag.**

**package** day7\_assign;

**class** FlagThread **extends** Thread {

**private** **volatile** **boolean** running = **true**;

**public** **void** stopThread() {

running = **false**;

}

**public** **void** run() {

**int** i = 0;

**while** (running) {

System.***out***.println("Thread running: " + i);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

i++;

}

System.***out***.println("Thread stopped.");

}

}

**public** **class** MT\_9 {

**public** **static** **void** main(String[] args) {

FlagThread thread = **new** FlagThread();

thread.start();

**try** {

Thread.*sleep*(5000);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

thread.stopThread();

}

}

**10. Create a program with multiple threads that access a shared counter without synchronization. Show the race condition.**

**package** day7\_assign;

**class** Counter {

**int** count = 0;

**public** **void** increment() {

count++;

}

**public** **int** getCount() {

**return** count;

}

}

**class** CounterThread **extends** Thread {

**private** Counter counter;

**public** CounterThread(Counter counter) {

**this**.counter=counter;

}

**public** **void** run() {

**for** (**int** i = 0;i<100000; i++) {

counter.increment();

}

}

}

**public** **class** MT\_10 {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

Counter counter = **new** Counter();

CounterThread thread1 = **new** CounterThread(counter);

CounterThread thread2 = **new** CounterThread(counter);

thread1.start();

thread2.start();

thread1.join();

thread2.join();

System.***out***.println("Actual count"+counter.getCount());

}

}

**11. solve the above problem using synchronized keyword to prevent race condition**.

**package** day7\_assign;

**class** Coun {

**int** count = 0;

**public** **synchronized** **void** increment() {

count++;

}

**public** **synchronized** **int** getCount() {

**return** count;

}

}

**class** CountT1 **extends** Thread {

**private** Coun counter;

**public** CountT1(Coun counter) {

**this**.counter = counter;

}

**public** **void** run() {

**for** (**int** i = 0;i<100000; i++) {

counter.increment();

}

}

}

**public** **class** MT\_11 {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

Counter counter = **new** Counter();

CounterThread thread1 = **new** CounterThread(counter);

CounterThread thread2 = **new** CounterThread(counter);

thread1.start();

thread2.start();

thread1.join();

thread2.join();

System.***out***.println("Actual count: " + counter.getCount());

}

}

**12. Write a Java program using synchronized block to ensure mutual exclusion.**

**package** day7\_assign;

**class** Count {

**int** count = 0;

**final** Object lock = **new** Object();

**public** **void** increment() {

**synchronized** (lock) {

count++;

}

}

**public** **int** getCount() {

**synchronized** (lock) {

**return** count;

}

}

}

**class** CounterT **extends** Thread {

**private** Count counter;

**public** CounterT(Count counter) {

**this**.counter = counter;

}

**public** **void** run() {

**for** (**int** i = 0; i < 100000; i++) {

counter.increment();

}

}

}

**public** **class** MT\_12 {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

Counter counter = **new** Counter();

CounterThread thread1 = **new** CounterThread(counter);

CounterThread thread2 = **new** CounterThread(counter);

thread1.start();

thread2.start();

thread1.join();

thread2.join();

System.***out***.println("Actual count: " + counter.getCount());

}

}

13. **Implement a BankAccount class accessed by multiple threads to deposit and withdraw money. Use synchronization**

**package** day7\_assign;

**class** BankAccount {

**private** **double** balance;

**final** Object lock = **new** Object();

**public** BankAccount(**double** initialBalance) {

**this**.balance=initialBalance;

}

**public** **void** deposit(**double** amount) {

**synchronized** (lock) {

balance += amount;

System.***out***.println("Deposited"+amount+"New balance"+balance);

}

}

**public** **void** withdraw(**double** amount) {

**synchronized** (lock) {

**if** (balance >= amount) {

balance -= amount;

System.***out***.println("Withdrawn"+amount+"New balance"+balance);

} **else** {

System.***out***.println("Insufficient");

}

}

}

**public** **double** getBalance() {

**synchronized** (lock) {

**return** balance;

}

}

}

**class** BankTransactionThread **extends** Thread {

**private** BankAccount account;

**private** **boolean** deposit;

**private** **double** amount;

**public** BankTransactionThread(BankAccount account, **boolean** deposit, **double** amount) {

**this**.account = account;

**this**.deposit = deposit;

**this**.amount = amount;

}

**public** **void** run() {

**if** (deposit) {

account.deposit(amount);

} **else** {

account.withdraw(amount);

}

}

}

**public** **class** MT\_13 {

**public** **static** **void** main(String[] args) {

BankAccount account = **new** BankAccount(1000.0);

BankTransactionThread thread1 = **new** BankTransactionThread(account, **true**, 500.0);

BankTransactionThread thread2 = **new** BankTransactionThread(account, **false**, 200.0);

BankTransactionThread thread3 = **new** BankTransactionThread(account, **true**, 300.0);

BankTransactionThread thread4 = **new** BankTransactionThread(account, **false**, 800.0);

thread1.start();

thread2.start();

thread3.start();

thread4.start();

**try** {

thread1.join();

thread2.join();

thread3.join();

thread4.join();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

System.***out***.println("Final balance"+account.getBalance());

}

}

14. **Create a Producer-Consumer problem using wait() and notify().**

**package** day7\_assign;

**class** SharedBuffer {

**private** **int** data;

**private** **boolean** available = **false**;

**public** **synchronized** **void** produce(**int** data) {

**while** (available) {

**try** {

wait();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

**this**.data = data;

available = **true**;

System.***out***.println("Produced: " + data);

notify();

}

**public** **synchronized** **int** consume() {

**while** (!available) {

**try** {

wait();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

**int** data = **this**.data;

available = **false**;

System.***out***.println("Consumed: " + data);

notify();

**return** data;

}

}

**class** ProducerThread **extends** Thread {

**private** SharedBuffer buffer;

**public** ProducerThread(SharedBuffer buffer) {

**this**.buffer = buffer;

}

**public** **void** run() {

**for** (**int** i = 1; i <= 5; i++) {

buffer.produce(i);

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

}

**class** ConsumerThread **extends** Thread {

**private** SharedBuffer buffer;

**public** ConsumerThread(SharedBuffer buffer) {

**this**.buffer = buffer;

}

**public** **void** run() {

**for** (**int** i = 1; i <= 5; i++) {

buffer.consume();

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

}

**public** **class** MT\_14 {

**public** **static** **void** main(String[] args) {

SharedBuffer buffer = **new** SharedBuffer();

ProducerThread producer = **new** ProducerThread(buffer);

ConsumerThread consumer = **new** ConsumerThread(buffer);

producer.start();

consumer.start();

**try** {

producer.join();

consumer.join();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

15. **Create a program where one thread prints A-Z and another prints 1-26 alternately.**

**package** day7\_assign;

**class** SharedPrinter {

**private** **boolean** printLetter = **true**;

**public** **synchronized** **void** printLetter(**char** letter) {

**while** (!printLetter) {

**try** {

wait();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

System.***out***.print(letter + " ");

printLetter = **false**;

notify();

}

**public** **synchronized** **void** printNumber(**int** number) {

**while** (printLetter) {

**try** {

wait();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

System.***out***.print(number + " ");

printLetter = **true**;

notify();

}

}

**class** LetterThread **extends** Thread {

**private** SharedPrinter printer;

**public** LetterThread(SharedPrinter printer) {

**this**.printer = printer;

}

**public** **void** run() {

**for** (**char** letter = 'A'; letter <= 'Z'; letter++) {

printer.printLetter(letter);

}

}

}

**class** NumT **extends** Thread {

**private** SharedPrinter printer;

**public** NumT(SharedPrinter printer) {

**this**.printer = printer;

}

**public** **void** run() {

**for** (**int** number = 1; number <= 26; number++) {

printer.printNumber(number);

}

}

}

**public** **class** MT\_15 {

**public** **static** **void** main(String[] args) {

SharedPrinter printer = **new** SharedPrinter();

LetterThread letterThread = **new** LetterThread(printer);

NumT numberThread = **new** NumT(printer);

letterThread.start();

numberThread.start();

**try** {

letterThread.join();

numberThread.join();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

16. **Write a program that demonstrates inter-thread communication using wait() and notifyAll().**

**package** day7\_assign;

**class** SharedResource {

**private** **boolean** available = **false**;

**public** **synchronized** **void** produce() {

**while** (available) {

**try** {

System.***out***.println("Producer waiting...");

wait();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

available = **true**;

System.***out***.println("Produced...");

notifyAll();

}

**public** **synchronized** **void** consume() {

**while** (!available) {

**try** {

System.***out***.println("Consumer waiting...");

wait();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

available = **false**;

System.***out***.println("Consumed...");

notifyAll();

}

}

**class** ProdT **extends** Thread {

**private** SharedResource resource;

**public** ProdT(SharedResource resource) {

**this**.resource = resource;

}

**public** **void** run() {

**for** (**int** i = 0; i < 5; i++) {

resource.produce();

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

}

**class** ConT **extends** Thread {

**private** SharedResource resource;

**public** ConT(SharedResource resource) {

**this**.resource = resource;

}

**public** **void** run() {

**for** (**int** i = 0; i < 5; i++) {

resource.consume();

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

}

**public** **class** MT\_16 {

**public** **static** **void** main(String[] args) {

SharedResource resource = **new** SharedResource();

ProdT producer = **new** ProdT(resource);

ConT consumer = **new** ConT(resource);

producer.start();

consumer.start();

**try** {

producer.join();

consumer.join();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

17. **Create a daemon thread that runs in background and prints time every second.**

**package** day7\_assign;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**class** TimeThread **extends** Thread {

**public** TimeThread() {

setDaemon(**true**);

}

**public** **void** run() {

SimpleDateFormat format = **new** SimpleDateFormat("HH:mm:ss");

**while** (**true**) {

System.***out***.println("Current Time: " + format.format(**new** Date()));

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

}

**public** **class** MT\_17 {

**public** **static** **void** main(String[] args) {

TimeThread timeThread = **new** TimeThread();

timeThread.start();

**try** {

Thread.*sleep*(10000);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

System.***out***.println("Main thread finished.");

}

}

18. **Demonstrate the use of Thread.isAlive() to check thread status.**

**package** day7\_assign;

**class** StatusThread **extends** Thread {

**public** **void** run() {

System.***out***.println("Thread is running");

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

System.***out***.println("Thread finished.");

}

}

**public** **class** MT\_18 {

**public** **static** **void** main(String[] args) {

StatusThread thread = **new** StatusThread();

System.***out***.println("Before starting thread"+thread.isAlive());

thread.start();

System.***out***.println("After starting thread"+thread.isAlive());

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

System.***out***.println("Thread status after 1 second"+thread.isAlive());

**try** {

thread.join();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

System.***out***.println("After thread finished"+thread.isAlive());

}

}

19. **Write a program to demonstrate thread group creation and management**

**package** day7\_assign;

**class** GroupThread **extends** Thread {

**public** GroupThread(ThreadGroup group, String name) {

**super**(group, name);

}

**public** **void** run() {

System.***out***.println("Thread"+Thread.*currentThread*().getName()+"is running");

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

System.***out***.println("Thread"+Thread.*currentThread*().getName()+"finished.");

}

}

**public** **class** MT\_19 {

**public** **static** **void** main(String[] args) {

ThreadGroup group = **new** ThreadGroup("MyThreadGroup");

GroupThread thread1 = **new** GroupThread(group,"Thread-1");

GroupThread thread2 = **new** GroupThread(group,"Thread-2");

GroupThread thread3 = **new** GroupThread(group,"Thread-3");

thread1.start();

thread2.start();

thread3.start();

System.***out***.println("Thread group name:"+group.getName());

System.***out***.println("Active thread count:"+group.activeCount());

**try** {

thread1.join();

thread2.join();

thread3.join();

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

System.***out***.println("All threads finished.");

}

}

20. **Create a thread that performs a simple task (like multiplication) and returns result using Callable and Future.**

**package** day7\_assign;

**import** java.util.concurrent.Callable;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**import** java.util.concurrent.Future;

**class** MultiplicationTask **implements** Callable<Integer> {

**private** **int** num1;

**private** **int** num2;

**public** MultiplicationTask(**int** num1, **int** num2) {

**this**.num1=num1;

**this**.num2=num2;

}

**public** Integer call() {

**return** num1 \* num2;

}

}

**public** **class** MT\_20 {

**public** **static** **void** main(String[] args) {

ExecutorService executor = Executors.*newSingleThreadExecutor*();

MultiplicationTask task = **new** MultiplicationTask(5, 10);

Future<Integer> future = executor.submit(task);

**try** {

System.***out***.println("Result"+future.get());

} **catch** (Exception e) {

System.***out***.println("An error occurred");

} **finally** {

executor.shutdown();

}

}

}