|  |
| --- |
| **package** crt;  **class** ThreadEx **extends** Thread  {  **public** **void** run()  {  System.***out***.println("Thread is running using thread extend");  }  }  **class** MyThreadEx **implements** Runnable  {  **public** **void** run()  {  System.***out***.println("Thread is running using thread implement");  }  }  **public** **class** MultiThreading {  **public** **static** **void** main(String args[])  {  ThreadEx te=**new** ThreadEx();  MyThreadEx mte=**new** MyThreadEx();  te.run();  mte.run();  }  } |
| **package** crt;  **class** Animal {  **void** makeSound() {  System.***out***.println("Generic animal sound");  }  }  **class** Dogg **extends** Animal {  @Override  **void** makeSound() {  System.***out***.println("Woof!");  }  **void** bark() {  System.***out***.println("Bark!");  }  }  **public** **class** Upcasting {  **public** **static** **void** main(String[] args) {  Dogg myDog = **new** Dogg();  Animal myAnimal = **new** Dogg(); // Upcasting: Dog object is now referenced as an Animal  Dogg my\_Dog = (Dogg) myAnimal; // Explicit downcasting  my\_Dog.makeSound();  my\_Dog.bark();  myAnimal.makeSound();  }  } |
| **package** crt;  **class** Node {  **int** key;  Node left, right;  Node(**int** item) {  key = item;  left = right = **null**;  }  }  **public** **class** BST\_Iterative {  **static** Node insert(Node root, **int** x) {  Node temp = **new** Node(x);  **if** (root == **null**) **return** temp;  Node parent = **null**;  Node curr = root;  **while** (curr != **null**) {  parent = curr;  **if** (x < curr.key)  curr = curr.left;  **else** **if** (x > curr.key)  curr = curr.right;  **else**  **return** root; // Key already exists  }  **if** (x < parent.key)  parent.left = temp;  **else**  parent.right = temp;  **return** root;  }  // Inorder Traversal  **static** **void** inorder(Node root) {  **if** (root != **null**) {  *inorder*(root.left);  System.***out***.print(root.key + " ");  *inorder*(root.right);  }  }  // Delete a node in BST  **static** Node deleteNode(Node root, **int** x) {  **if** (root == **null**) **return** root;  **if** (x < root.key)  root.left = *deleteNode*(root.left, x);  **else** **if** (x > root.key)  root.right = *deleteNode*(root.right, x);  **else** {  // Node with one child or no child  **if** (root.left == **null**)  **return** root.right;  **else** **if** (root.right == **null**)  **return** root.left;  // Node with two children: Get inorder successor  Node succ = *getSuccessor*(root.right);  root.key = succ.key;  root.right = *deleteNode*(root.right, succ.key);  }  **return** root;  }  // Get inorder successor (smallest in the right subtree)  **static** Node getSuccessor(Node curr) {  **while** (curr.left != **null**)  curr = curr.left;  **return** curr;  }  // Main method  **public** **static** **void** main(String[] args) {  Node root = **null**;  // Insert nodes  root = *insert*(root, 10);  *insert*(root, 5);  *insert*(root, 15);  *insert*(root, 12);  *insert*(root, 18);  System.***out***.print("Inorder before deletion: ");  *inorder*(root);  **int** x = 15;  root = *deleteNode*(root, x);  System.***out***.print("\nInorder after deleting " + x + ": ");  *inorder*(root);  }  } |
| **package** crt;  **class** Node {  **int** key;  Node left, right;  **public** Node(**int** item) {  key = item;  left = right = **null**;  }  }  **public** **class** Bst\_rec {  // Insert method  **static** Node insert(Node root, **int** key) {  // If the tree is empty, return a new node  **if** (root == **null**) {  **return** **new** Node(key);  }  // If key is already present, return the node  **if** (root.key == key) {  **return** root;  }  // Otherwise, recur down the tree  **if** (key < root.key) {  root.left = *insert*(root.left, key);  } **else** {  root.right = *insert*(root.right, key);  }  // Return the (unchanged) node pointer  **return** root;  }  // Utility function for inorder traversal  **static** **void** inorder(Node root) {  **if** (root != **null**) {  *inorder*(root.left);  System.***out***.print(root.key + " ");  *inorder*(root.right);  }  }  // Driver method  **public** **static** **void** main(String[] args) {  Node root = **null**;  root = *insert*(root, 50);  *insert*(root, 30);  *insert*(root, 20);  *insert*(root, 40);  *insert*(root, 70);  *insert*(root, 60);  *insert*(root, 80);  System.***out***.println("Inorder traversal of BST:");  *inorder*(root);  }  } |
| **package** crt;  **class** MyThread **extends** Thread {  **volatile** **boolean** running = **true**;  **public** **void** run() {  **while** (running) {  System.***out***.println("Running...");  }  System.***out***.println("Stopped.");  }  **public** **void** stopThread() {  running = **false**;  }  }  **public** **class** ThreadMain {  **public** **static** **void** main(String[] args) **throws** InterruptedException {  MyThread t = **new** MyThread();  t.start();  Thread.*sleep*(10);  t.stopThread(); // Gracefully stop  t.join(); // Wait until it finishes  }  } |
| **package** crt;  **class** MyDaemon **extends** Thread {  **public** **void** run() {  **while** (**true**) {  System.***out***.println("Daemon thread running...");  **try** {  Thread.*sleep*(500);  } **catch** (InterruptedException e) {  System.***out***.println(e);  }  }  }  }  **public** **class** DaemonThread {  **public** **static** **void** main(String[] args) {  MyDaemon t = **new** MyDaemon();  t.setDaemon(**true**); // Must be called before start()  t.start();  System.***out***.println("Main thread sleeping for 2 seconds...");  **try** {  Thread.*sleep*(2000);  } **catch** (InterruptedException e) {  System.***out***.println(e);  }  System.***out***.println("Main thread ends — JVM exits, daemon dies.");  }  } |
|  |
|  |