Code Fragment 10.9

```
/** Realization of a dictionary by means of a red-black tree. */
public class RBTree<K,V>
  extends BinarySearchTree<K,V> implements Dictionary<K,V> {
  public RBTree() { super(); }
  public RBTree(Comparator<K> C) { super(C); }
  /** Nested class for the nodes of a red-black tree */
  protected static class RBNode<K,V> extends BTNode<Entry<K,V>> {
    protected boolean isRed;  // we add a color field to a BTNode
    RBNode() {/* default constructor */}
    /** Preferred constructor */
    RBNode(Entry<K,V> element, BTPosition<Entry<K,V>> parent,
           BTPosition<Entry<K,V>> left, BTPosition<Entry<K,V>> right) {
      super(element, parent, left, right);
      isRed = false;
    }
    public boolean isRed()  {return isRed;}
    public void makeRed()  {isRed = true;}
    public void makeBlack()  {isRed = false;}
    public void setColor(boolean color)  {isRed = color;}
  }
```

Code Fragment 10.10

```
/** Creates a new tree node. */
 protected BTPosition<Entry<K,V>> createNode(Entry<K,V> element,
    BTPosition<Entry<K,V>> parent, BTPosition<Entry<K,V>> left,
    BTPosition<Entry<K,V>> right) {
  return new RBNode<K,V>(element,parent,left,right); // a red-black node
 }
 public Entry<K,V> insert(K k, V x) throws InvalidKeyException {
  Entry<K,V> toReturn = super.insert(k, x);
  Position<Entry<K,V>> posZ = actionPos; // start at the insertion position
  setRed(posZ);
  if (isRoot(posZ))
    setBlack(posZ);
  else
    remedyDoubleRed(posZ); // fix a double-red color violation
  return toReturn;
 }
 protected void remedyDoubleRed(Position<Entry<K,V>> posZ) {
  Position<Entry<K,V>> posV = parent(posZ);
  if (isRoot(posV))
    return;
  if (!isPosRed(posV))
    return;
  // we have a double red: posZ and posV
  if (!isPosRed(sibling(posV))) { // Case 1: trinode restructuring
    posV = restructure(posZ);
    setBlack(posV);
    setRed(left(posV));
    setRed(right(posV));
  }
  else { // Case 2: recoloring
    setBlack(posV);
    setBlack(sibling(posV));
    Position<Entry<K,V>> posU = parent(posV);
    if (isRoot(posU))
      return;
    setRed(posU);
    remedyDoubleRed(posU);
  }
 }
```

Code Fragment 10.11

```
public Entry<K,V> remove(Entry<K,V> ent) throws InvalidEntryException {
   Entry<K,V> toReturn = super.remove(ent);
   Position<Entry<K,V>> posR = actionPos;
   if (toReturn != null) {
     if (wasParentRed(posR) || isRoot(posR) || isPosRed(posR))
         setBlack(posR);
     else
         remedyDoubleBlack(posR);
   }
   return toReturn;
 }
 protected void remedyDoubleBlack(Position<Entry<K,V>> posR) {
   Position<Entry<K,V>> posX, posY, posZ;
   boolean oldColor;
   posX = parent(posR);
   posY = sibling(posR);
   if (!isPosRed(posY))  {
     posZ = redChild(posY);
     if (hasRedChild(posY))  { // Case 1: trinode restructuring
         oldColor = isPosRed(posX);
         posZ = restructure(posZ);
         setColor(posZ, oldColor);
         setBlack(posR);
         setBlack(left(posZ));
         setBlack(right(posZ));
         return;
     }
     setBlack(posR);
     setRed(posY);
     if (!isPosRed(posX))  { // Case 2: recoloring
       if (!isRoot(posX))
           remedyDoubleBlack(posX);
         return;
     }
     setBlack(posX);
     return;
   } // Case 3: adjustment
   if (posY == right(posX)) posZ = right(posY);
   else posZ = left(posY);
   restructure(posZ);
   setBlack(posY);
   setRed(posX);
   remedyDoubleBlack(posR);
 }
```