```java
import java.util.*;

class TreeNode {
    Object key;
    TreeNode left, right;

    public TreeNode(Object key) {
        this.key = key;
        left = right = null;
    }
}

class BinarySearchTree {
    private TreeNode root;

    public BinarySearchTree() {
        root = null;
    }

    public void insert(Object key) {
        root = insertRec(root, key);
    }

    private TreeNode insertRec(TreeNode root, Object key) {
        if (root == null) {
            root = new TreeNode(key);
            return root;
        }

        if (key.equals(root.key))
            return root;

        if (compareKeys(key, root.key) < 0)
            root.left = insertRec(root.left, key);
        else
            root.right = insertRec(root.right, key);

        return root;
    }

    private int compareKeys(Object key1, Object key2) {
        if (key1 instanceof Integer) {
            return ((Integer) key1).compareTo((Integer) key2);
        } else if (key1 instanceof String) {
            return ((String) key1).compareTo((String) key2);
        } else if (key1 instanceof Double) {
            return ((Double) key1).compareTo((Double) key2);
        } else {
            throw new IllegalArgumentException("Unsupported key type");
        }
    }

    public PreorderIterator getPreorderIterator() {
        return new PreorderIterator(root);
    }

    public class PreorderIterator {
        private Stack<TreeNode> stack;

        public PreorderIterator(TreeNode root) {
            stack = new Stack<>();
            if (root != null)
                stack.push(root);
        }

        public boolean hasNext() {
            return !stack.isEmpty();
        }

        public TreeNode next() {
            if (!hasNext())
                throw new NoSuchElementException("No more elements in the iterator");

            TreeNode node = stack.pop();
            if (node.right != null)
                stack.push(node.right);
            if (node.left != null)
                stack.push(node.left);

            return node;
        }
    }
}
```

```java
class BinarySearchTreeFactory {
    public static BinarySearchTree readTree(Scanner scanner) throws UnexpectedInputException {
        if (!scanner.hasNextLine())
            throw new UnexpectedInputException("Input is empty");

        String keyTypeStr = scanner.nextLine().trim();
        Object[] keys = readKeys(scanner, keyTypeStr);

        BinarySearchTree tree = new BinarySearchTree();
        for (Object key : keys) {
            tree.insert(key);
        }
        return tree;
    }

    private static Object[] readKeys(Scanner scanner, String keyTypeStr) throws UnexpectedInputException {
        List<Object> keys = new ArrayList<>();
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine().trim();
            if (line.isEmpty())
                break;

            Object key = parseKey(line, keyTypeStr);
            keys.add(key);
        }
        return keys.toArray(new Object[0]);
    }

    private static Object parseKey(String keyStr, String keyTypeStr) throws UnexpectedInputException {
        try {
            switch (keyTypeStr) {
                case "Integer":
                    return Integer.parseInt(keyStr);
                case "String":
                    return keyStr;
                case "Double":
                    return Double.parseDouble(keyStr);
                default:
                    throw new UnexpectedInputException("Unsupported key type: " + keyTypeStr);
            }
        } catch (NumberFormatException e) {
            throw new UnexpectedInputException("Invalid key format: " + keyStr);
        }
    }
}

class UnexpectedInputException extends Exception {
    public UnexpectedInputException(String message) {
        super(message);
    }
}

public class Main {
    public static void main(String[] args) {
        try {
            Scanner inpFile = new Scanner(System.in);
            BinarySearchTree t = BinarySearchTreeFactory.readTree(inpFile);
            BinarySearchTree.PreorderIterator it = t.getPreorderIterator();
            while (it.hasNext()) {
                TreeNode node = it.next();
                System.out.println(node.key);
            }
            it.next(); // one more next(), to trigger that exception
        } catch (UnexpectedInputException e) {
            System.out.println("Unexpected input: " + e.getMessage());
        } catch (NoSuchElementException e) {
            System.out.println("Iterator exception: " + e.getMessage());
        }
    }
}
```