Wzorce Projektowe, lab. 11

Temat: Praktyczne zastosowanie wzorca **Iterator**.

Twoje zadanie to zaimplementowanie struktury drzewa BST oraz metody na dodawanie elementów. Następnie masz stworzyć kilka iteratorów pozwalających na uzyskiwanie kolejnych elementów z tego drzewa w odpowiedni sposób.

1. Zacznij od utworzenia interfejsu `Iterator<T>` z metodami `boolean hasNext()` oraz `T next()`.
2. Utwórz klasę opisującą pojedynczy element w drzewie `BSTElement<T>` z polem `data` typu **T** oraz referencjami na trzy inne elementy: `leftChild, rightChild, parent` – będą one niezbędne do prawidłowego poruszania się po drzewie.
3. Utwórz abstrakcyjną klasę `BSTIterator<T>` implementującą interfejs `Iterator<T>`. Powinna ona zawierać w sobie listę typu `ArrayList<BSTElement<T>>`, która będzie przechowywała referencje na kolejne elementy drzewa, posortowane w odpowiedni sposób. Oprócz tego ma się tam znaleźć także numer aktualnego indexu na liście.
4. W powyższej klasie:
    - zadeklaruj abstrakcyjną metodę do wypełniania listy, która będzie przyjmowała w argumencie element drzewa, ale nie implementuj jej. Zrobisz to w konkretnych klasach iteratorów określających kolejność wypełniania elementami: preorder, inorder lub postorder.
    - stwórz metodę do resetowania listy, która przyjmie w argumencie korzeń drzewa. Ma ona: czyścić listę, wywoływać metodę do wypełnienia listy (argumentem będzie korzeń) i zerować aktualny index.
    - stwórz konstruktor `protected BSTIterator(BSTElement<T> root)`, który zainicjalizuje listę i zresetuje ją przekazując w argumencie korzeń.
    - nadpisz metody `hasNext()` i `next()`: pierwsza zwróci *true*, jeśli aktualny index nie przekracza ilości elementów na liście, a druga zwróci element z listy z aktualnym indexem i zwiększy index o 1.
5. Utwórz klasę `IntBinarySearchTree`, która będzie już konkretną implementacją drzewa BST dla typu Integer. Ma ona w sobie mieć referencję na korzeń drzewa typu `BSTElement<Integer>`, a także listę utworzonych iteratorów (np. ArrayListę). Oprócz tego ma się tam znaleźć metoda `void addElement(int data)`, która ma dodawać element `BSTElement<Integer>` do drzewa w odpowiedni sposób (lewe dziecko < rodzica, prawe dziecko >= rodzicowi). Algorytmy realizujące tę operację są dostępne w internecie. Pamiętaj też, że dodanie elementu do drzewa powinno wiązać się ze zresetowaniem wszystkich iteratorów - właśnie w tym celu mamy w klasie listę z nimi.
6. Wewnątrz klasy `IntBinarySearchTree` utwórz trzy prywatne klasy dziedziczące z `BSTIterator<Integer>`: `PreorderIterator, InorderIterator, PostorderIterator`. Każda z nich ma w konstruktorze odwoływać się do konstruktora nadklasy przekazując mu referencję do korzenia drzewa z klasy (jest on dostępny bezpośrednio, ponieważ klasy te są wewnątrz klasy `IntBinarySearchTree`), a także nadpisywać metodę do wypełniania listy. Dzięki przekazywaniu do niej w argumencie konkretnego elementu

możesz zastosować tu rekurencję. Przykładowo iterator inorder najpierw odwoła się rekurencyjnie do lewego poddrzewa, potem doda do listy wartość z aktualnego elementu, a potem odwoła się rekurencyjnie do prawego poddrzewa. Pozostałe iteratory będą miały jedynie zmienioną kolejność odwołań. W ten sposób Twoja lista od razu będzie posortowana w odpowiedni sposób.

7. Klasa `IntBinarySearchTree` ma udostępniać trzy publiczne metody: `public Iterator<Integer> preorderIterator()`, `public Iterator<Integer> inorderIterator()`, `public Iterator<Integer> postorderIterator()`. Każda z nich ma utworzyć nowy obiekt odpowiedniego iteratora, dodać go do listy iteratorów i zwrócić.
8. W funkcji *main()* utwórz nowe drzewo `IntBinarySearchTree`. Za pomocą obiektu typu `Scanner` wczytaj z konsoli przykładową listę liczb i wypełnij nimi drzewo. Następnie utwórz trzy iteratory: jeden preorder, drugi inorder i trzeci postorder. Każdy z nich ma wypisać kolejno wszystkie elementy z drzewa. Następnie znowu wczytaj z konsoli grupę liczb, dodaj do drzewa i ponownie wypisz wszystkie elementy drzewa za pomocą utworzonych wcześniej iteratorów.

UWAGI:

- Zauważ, że na etapie tworzenia interfejsu i klasy abstrakcyjnej `BSTIterator<T>` posługujemy się typem sparametryzowanym T. Konkretny typ jest nadawany dopiero później.