Assignment #5

Implementing Heap File and Sorted File for Data Storage

Assignment Overview

- **Goal**: Design and implement a Heap File & a Sorted File
- The following Java source files are included in `comp332-assignment5.zip`:
  - HeapFile.java
  - SortedFile.java

  } Modify these files

  - HeapFileBasicTest.java
  - SortedFileBasicTest.java
  - PerformanceTest.java

  } Use these files for testing

  - PageDirectory.java
  - PageInfo.java
  - Page.java
  - Record.java

  } DO NOT modify these files

Task: Implement Core Functions

- Implement the following four functions in HeapFile.java and SortedFile.java files:
  1) insertRecord
  2) searchRecord
  3) deleteRecord
  4) rangeSearch

```java
/**
 * Inserts a record into the heap file.
 * Allocates a new page if no free slots are available.
 *
 * @param record The record to insert.
 * @throws IOException If an I/O error occurs during the operation.
 */
public void insertRecord(Record record) throws IOException {  3 usages
    // TODO: Implement this function.
    return;
}
```

- Each function is marked with the comment:
  - // *TODO: Implement this function.*

Task: Implement Packing Strategies

- We use *different strategies* for handling packing in the Heap File and Sorted File to explore their trade-offs
- **Heap File**
  - Do not perform packing (prioritizing efficient delete operations)
  - Delete operation: Simply mark the slot as unused without reorganizing the page (i.e., no packing).
- **Sorted File**
  - Perform packing (prioritizing efficient use of storage capacity)
  - Delete operation: Remove the record and shift remaining records to eliminate gaps in the page (i.e., perform packing).

Task: Writing a Brief Report

- After completing the implementation, perform a simple experiment with the Heap File and Sorted File.
  - You can simply use the provided PerformanceTest.java file for performance measurement.
- Write a brief report that:
  - Explains your source code for Heap File and Sorted File.
  - Analyzes the performance of your Heap File and Sorted File.
- Please include a graph of the experimental results in your report.

How to Test Your Code

- Use the following Java files for testing:
  - **HeapFileBasicTest.java**
    - Tests basic functionality of the Heap File, including insert, delete, search, and range search.
  - **SortedFileBasicTest.java**
    - Tests basic functionality of the Sorted File.
  - **PerformanceTest.java**
    - Compares the performance of Heap File and Sorted File for operations such as insert, search, and range search.
- Reference outputs are provided in the `comp332-assignment5.zip` file:
  - HeapFileBasicTest_ReferenceOutput.txt
  - SortedFileBasicTest_ReferenceOutput.txt

Submission

- **Deadline**: **23:59:59 on December 5th, 2024 (12/5/2024)**
- **Submission Instructions**:
  - Submit a **single** `.zip` file (`assignment5-`*Your_Student_ID*`.zip`)
  - The `.zip` file should contain the following three files:

    (1) HeapFile.java

    (2) SortedFile.java

    (3) Report.pdf

Appendix

Environment Setup

Development Environment Setup

- We recommend using **IntelliJ IDEA** for the assignment.
- You have two options:
  - **IntelliJ IDEA Ultimate**:
    - Download from: https://www.jetbrains.com/idea/download
    - Apply for a Free Educational License using your university email:
      - https://www.jetbrains.com/community/education/#students

  - **IntelliJ IDEA Community Edition**:
    - This is a free version with limited features
    - Download from: https://www.jetbrains.com/idea/download/download-thanks.html?platform=windows&code=IIC

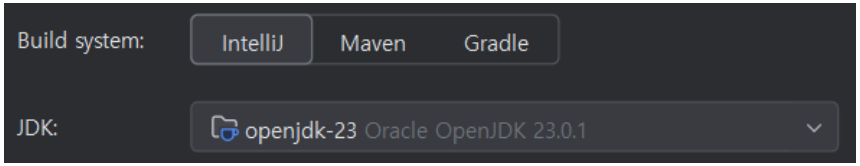
Development Environment Setup

- **Other IDEs**:
  - Or you can use alternative IDEs such as Eclipse.

- **Command-line approach**:
  - If you prefer, you can simply use a terminal and run Java command directly.

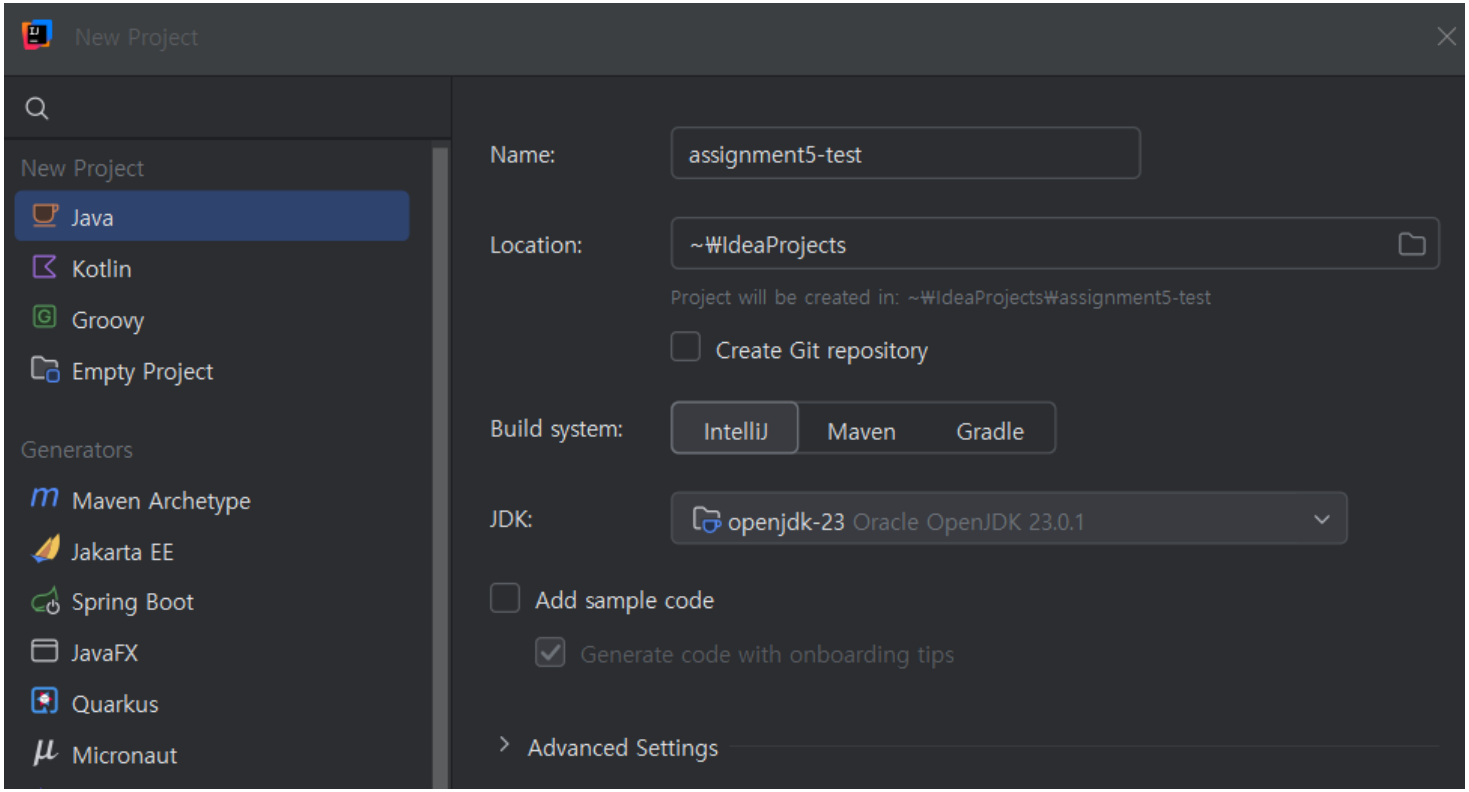
```
ubuntu at cloud in ~/assignment5
→ javac *.java

ubuntu at cloud in ~/assignment5
→ java HeapFileBasicTest

HeapFile Pages:
```

- **Java version**
  - Assignment has been tested using Oracle OpenJDK 23.0.1



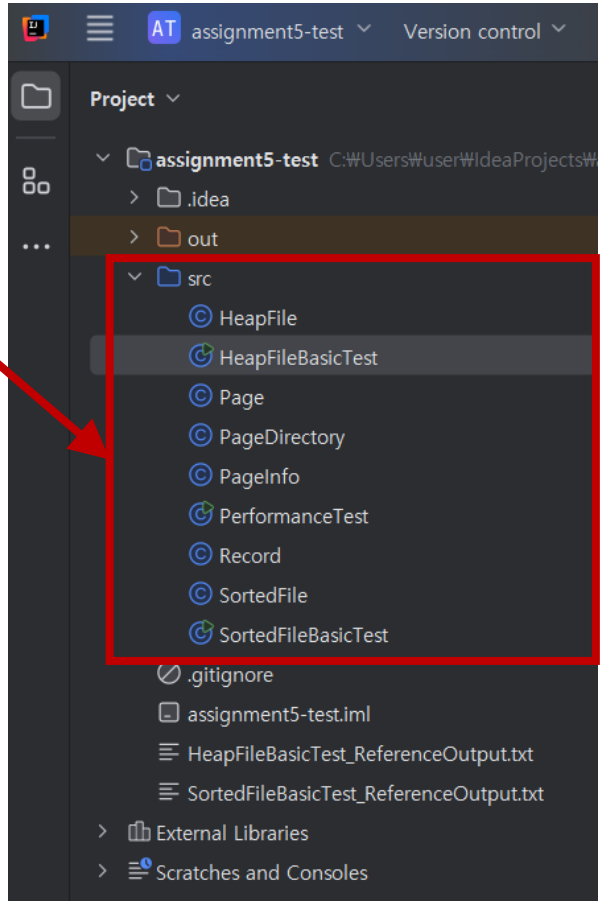
Environment Setup with IntelliJ

- Create new project



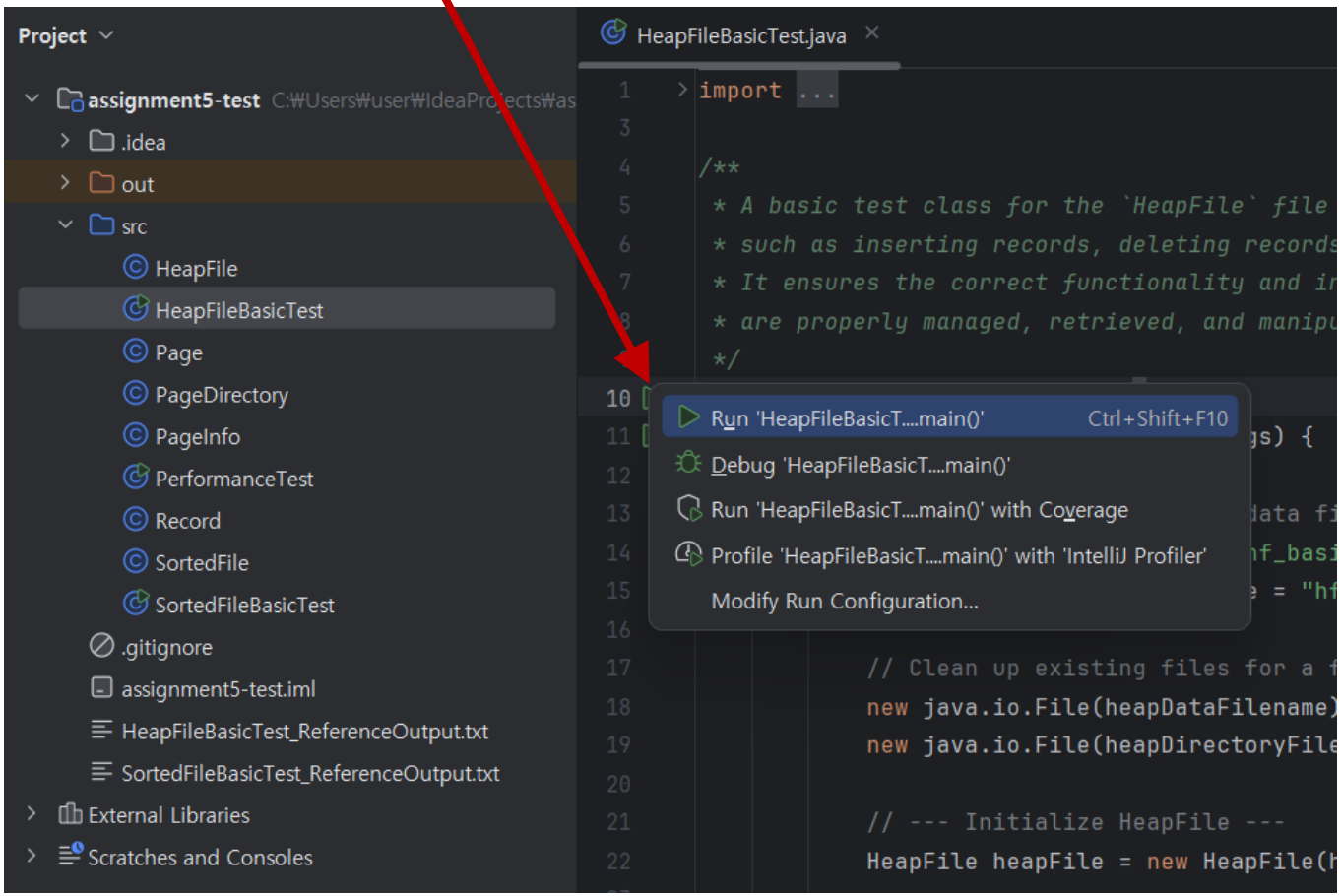
Environment Setup with IntelliJ

- Copy the source code files to the "src" directory.



Environment Setup with IntelliJ

- Run your code (Click ▶ button or Ctrl+Shift+F10)



Appendix

Technical Details and Implementation Guidelines

Record Format

- For this assignment, we are using a simplified record structure:
  - Fixed-format records
  - Fixed-length records
- The record structure consists of:
  - Key: 4-byte integer
  - Data: 250-byte fixed-length string

```
public class Record {  54 usages
    private int key; // Unique identifier for the record  3 usages
    private String data; // Fixed-length string (250 bytes)  4 usages

    public static final int DATA_SIZE = 250; // Fixed size of data in
    public static final int RECORD_SIZE = Integer.BYTES + DATA_SIZE;
```

Page Layout

- Page Size: 4 KB (4096 bytes)
- Header Type: Bitmap header
- Header Function: Tracks currently used slots
  - A set bit (1) indicates an occupied slot
  - An unset bit (0) indicates an available slot
  - See isSlotUsed, setSlotUsed functions in the Page class
- Record Capacity: Maximum of 16 records per page

```
public class Page {  28 usages
    private byte[] header; // Bitmap to track used slots  7 usages
    private Record[] records; // Array to store records in the page  8 usages

    public static final int PAGE_SIZE = 4096; // 4KB page size  4 usages
    public static final int RECORD_SIZE = Record.RECORD_SIZE;  2 usages
    public static final int SLOT_COUNT = 16; // Number of record slots  8 usag
    public static final int HEADER_SIZE = (int) Math.ceil(SLOT_COUNT / 8.0);
```

Page Directory

- We employ a page directory to efficiently track free slots in each page:
  - Directory Entry: (file offset, # of free slots) for each page
    - PageInfo class represents a directory entry in our implementation

```
public class PageInfo implements Serializable {
    private long offset;     // Starting offset
    private int freeSlots;   // Number of free
```

  - Refer to PageInfo.java and PageDirectory.java files for details
- For simplicity, we store the directory in a separate file:
  - Example: `hf_basic_test.pd`

```
public class HeapFileBasicTest {
    public static void main(String[] args) {
        try {
            // Filenames for HeapFile (data file and directory file)
            String heapDataFilename = "hf_basic_test.dat";
            String heapDirectoryFilename = "hf_basic_test.pd";
```

Expected Behavior of Heap File

- **Insert Operation**
  - Perform a linear scan from the first page
  - Find the first available free slot
  - If all pages are full, allocate a new page
- **Search Operation**
  - Conduct a linear scan from the first page to the last page
- **Delete Operation**
  - Locate the target record
  - Mark the slot as unused in the page header's bitmap
    - Note: we do not perform packing for Heap File

Expected Behavior of Sorted File

- **Insert Operation**
  - Locate the correct position in the sorted data
  - Insert the record, maintaining the sort order
  - If the page is full, create space by reorganizing or allocating a new page
  - Note: This operation ensures data remains sorted for efficient searching
- **Search Operation**
  - Perform a binary search to efficiently locate the target record
- **Delete Operation**
  - Find the target record
  - Remove the record
  - Shift remaining records to eliminate gaps in the page (perform packing)
    - Note: Packing is required for Sorted File