Simple graphs are only used to describe the structure of data. To use graphs, we must define a graph object in the simplest form.

To use Graphs, we must define a Graph object in this simplest form:

```
Graph { new Graph() }
```

To use Graphs, we must define a Graph object in the simplest form.

```
Graph { new Graph() }
```

The Graph class:

```
class Graph {
  // The Graph class defines the fundamental structure of a graph.
  // It includes methods for adding vertices, edges, and performing
  // various graph algorithms.
}
```

Here is a list of the key methods and the four common containers that Graph defines:

- Graph class methods:
  - `addVertex()`: Adds a new vertex to the graph.
  - `addEdge()`: Adds an edge between two vertices.
  - `getVertices()`: Returns a list of all vertices in the graph.
  - `getEdges()`: Returns a list of all edges in the graph.
  - `hasVertex()`: Checks if a vertex exists in the graph.
  - `hasEdge()`: Checks if an edge exists between two vertices.
  - `toString()`: Returns a string representation of the graph.

These methods allow for the manipulation and querying of the graph structure.

The Graph class:

```
class Graph {
  // The Graph class defines the fundamental structure of a graph.
  // It includes methods for adding vertices, edges, and performing
  // various graph algorithms.
}
```

Here is a list of the key methods and the four common containers that Graph defines:

- Graph class methods:
  - `addVertex()`: Adds a new vertex to the graph.
  - `addEdge()`: Adds an edge between two vertices.
  - `getVertices()`: Returns a list of all vertices in the graph.
  - `getEdges()`: Returns a list of all edges in the graph.
  - `hasVertex()`: Checks if a vertex exists in the graph.
  - `hasEdge()`: Checks if an edge exists between two vertices.
  - `toString()`: Returns a string representation of the graph.

These methods allow for the manipulation and querying of the graph structure.

The Graph class:

```
class Graph {
  // The Graph class defines the fundamental structure of a graph.
  // It includes methods for adding vertices, edges, and performing
  // various graph algorithms.
}
```

Here is a list of the key methods and the four common containers that Graph defines:

- Graph class methods:
  - `addVertex()`: Adds a new vertex to the graph.
  - `addEdge()`: Adds an edge between two vertices.
  - `getVertices()`: Returns a list of all vertices in the graph.
  - `getEdges()`: Returns a list of all edges in the graph.
  - `hasVertex()`: Checks if a vertex exists in the graph.
  - `hasEdge()`: Checks if an edge exists between two vertices.
  - `toString()`: Returns a string representation of the graph.

These methods allow for the manipulation and querying of the graph structure.

The Graph class:

```
class Graph {
  // The Graph class defines the fundamental structure of a graph.
  // It includes methods for adding vertices, edges, and performing
  // various graph algorithms.
}
```

Here is a list of the key methods and the four common containers that Graph defines:

- Graph class methods:
  - `addVertex()`: Adds a new vertex to the graph.
  - `addEdge()`: Adds an edge between two vertices.
  - `getVertices()`: Returns a list of all vertices in the graph.
  - `getEdges()`: Returns a list of all edges in the graph.
  - `hasVertex()`: Checks if a vertex exists in the graph.
  - `hasEdge()`: Checks if an edge exists between two vertices.
  - `toString()`: Returns a string representation of the graph.

These methods allow for the manipulation and querying of the graph structure.

The Graph class:

```
class Graph {
  // The Graph class defines the fundamental structure of a graph.
  // It includes methods for adding vertices, edges, and performing
  // various graph algorithms.
}
```

Here is a list of the key methods and the four common containers that Graph defines:

- Graph class methods:
  - `addVertex()`: Adds a new vertex to the graph.
  - `addEdge()`: Adds an edge between two vertices.
  - `getVertices()`: Returns a list of all vertices in the graph.
  - `getEdges()`: Returns a list of all edges in the graph.
  - `hasVertex()`: Checks if a vertex exists in the graph.
  - `hasEdge()`: Checks if an edge exists between two vertices.
  - `toString()`: Returns a string representation of the graph.

These methods allow for the manipulation and querying of the graph structure.
The graph, class 233

Advanced Features

The graph is used to display data.

The graph provides a visual representation of data, which is useful for analyzing trends and patterns in the data. The x-axis represents the horizontal axis, while the y-axis represents the vertical axis. Each point on the graph corresponds to a specific data point, allowing for easy comparison and analysis of the data.

The graph can be used to identify trends, outliers, and patterns in the data. By examining the graph, one can quickly identify any anomalies or unusual observations, which can be further investigated to determine their cause. Additionally, the graph can be used to make predictions and forecast future trends based on the data.

The graph is a powerful tool for data analysis and can be used in various fields, such as business, economics, science, and engineering, to name a few. It is an essential tool for anyone working with data, as it provides a visual representation of the data, making it easier to understand and communicate findings.

In conclusion, graphs are an essential tool for data analysis and can be used to identify trends, patterns, and anomalies in the data. By using graphs, one can make informed decisions and take appropriate actions based on the data analysis.
null

---

null

---

null
Figure 6.13

The graph is shown in Figure 6.13. For the data shown above, the graph is as follows. The graph class 237.