1. In Java, how do you utilize Comparator and Comparable? Give example.

In Java, Comparable and Comparator are interfaces used to define the ordering of objects.

Comparable is used when an object has a natural ordering. The class whose objects need to be compared implements the Comparable interface and overrides the compareTo() method.

Example:
```
class Student implements Comparable {
int marks;
public int compareTo(Student s) {
return this.marks - s.marks;
}
}
```

```
Collections.sort(studentList);
```

Comparator is used when we want multiple or custom sorting logics. A separate class implements Comparator and overrides the compare() method.

Example:
```
class MarksComparator implements Comparator {
public int compare(Student s1, Student s2) {
return s2.marks - s1.marks;
}
}
```

```
Collections.sort(studentList, new MarksComparator());
```

Thus, Comparable defines natural ordering, while Comparator provides external and flexible sorting logic.

2. Differentiate between Comparable and Comparator interfaces.

Comparable:
- Located in java.lang package
- Used for natural ordering
- Class must implement Comparable
- Method used: compareTo(Object o)
- Only one sorting sequence is possible

Comparator:
- Located in java.util package
- Used for custom ordering
- Separate class implements Comparator
- Method used: compare(Object o1, Object o2)
- Multiple sorting sequences are possible

Hence, Comparable is internal to the class, while Comparator is external and more flexible.

3. With example define the concept of lambda expressions. Explain the different types of lambda expressions.

Lambda expressions were introduced in Java 8 to represent anonymous functions. They allow us to write concise and readable code, especially when working with functional interfaces.

Syntax:
(parameters) -> expression or block

Example:
Runnable r = () -> System.out.println("Hello");

Types of Lambda Expressions:
1. Lambda with no parameter:
() -> System.out.println("Hello");

2. Lambda with one parameter:
(a) -> a * a

3. Lambda with multiple parameters:
(a, b) -> a + b

4. Lambda with return statement:
(a, b) -> { return a + b; }

Lambda expressions reduce boilerplate code and support functional programming concepts in Java.

4. Write a program to implement lambda expression accepting two parameters.

Program:

```
interface Add {
int sum(int a, int b);
}

public class LambdaDemo {
public static void main(String[] args) {
Add obj = (a, b) -> a + b;
System.out.println("Sum = " + obj.sum(10, 20));
}
}
```

Explanation:
Here, lambda expression accepts two parameters a and b and returns their sum using a functional interface.

5. What is the use of Properties class? Describe the various constructors used in Properties class.

The Properties class in Java is a subclass of Hashtable used to store key-value pairs where both keys and values are strings. It is mainly used to read configuration data from files.

Uses:
- Store configuration settings
- Read data from .properties files
- Persist application settings

Constructors:
1. Properties():
Creates an empty properties list.

2. Properties(Properties defaults):
Creates a properties object with default values.

Example:
```
Properties p = new Properties();
Properties p2 = new Properties(p);
```

Thus, Properties class helps in managing application configuration efficiently.

6. With example elucidate the various methods of Properties class.

Important methods of Properties class:

1. setProperty(String key, String value):
```
p.setProperty("user", "admin");
```

2. getProperty(String key):
```
String user = p.getProperty("user");
```

3. load(InputStream in):
```
p.load(new FileInputStream("config.properties"));
```

4. store(OutputStream out, String comments):
```
p.store(new FileOutputStream("config.properties"), "App Config");
```

5. getProperty(String key, String defaultValue):
p.getProperty("port", "8080");

Example:
Properties p = new Properties();
p.setProperty("language", "Java");
System.out.println(p.getProperty("language"));

These methods help in reading, writing, and managing configuration data.