- In this session, you will learn to:
  - Use abstract classes
  - Use the static and final keywords

- When sibling classes have a common method, it should be placed in a parent class.
- Abstract parent class:
  - Should contain the common methods of the sibling classes
  - Methods can be overridden with a specialized implementation
- The following embedded Word document shows a subclass that implements the methods of an abstract parent class.

- Abstract classes:
  - Declared using the `abstract` modifier, as shown in the following code snippet:

    ```
    public abstract class ElectronicDevice { }
    ```
  - Can be subclassed, as shown in the following code snippet:

    ```
    public class Television extends
    ElectronicDevice { }
    ```
  - Cannot be instantiated, as shown in the following code snippet:

    ```
    ElectronicDevice dev = new ElectronicDevice();
    ```

    Error
  - Can be used as a reference type

- An abstract method:
  - Can be declared by using the `abstract` modifier, as shown in the following code snippet:

```
public abstract class ElectronicDevice
{
public abstract void turnOn();
public abstract void turnOff();
}
```

No braces

  - Cannot have a body
  - Must be declared in an abstract class
  - Can be overridden in the subclasses
- A child class that inherits an abstract method, inherits the method signature but not the implementation.

- Rules for using abstract classes and methods:
  - An abstract class may have any number of abstract and non abstract methods.
  - While inheriting from an abstract class:
    - Declare the child class as abstract.

      Or
    - Override all abstract methods inherited from the parent class.

- The `static` modifier is used to declare fields and methods as class-level resources.
- Static class members:
  - Used when objects of the same type need to share fields
  - Used when a problem can be solved without using objects
  - Used without object instances
  - Should not be used to bypass the object orientation

- Static methods:
  - Can be called without an instance of a class
  - Also called class methods
  - Useful for APIs that are not object-oriented
  - Can be used instead of constructors to object initialization
  - Cannot access non-static members of the class
  - Can be hidden in the subclasses, but cannot be overridden
  - Can be used to retrieve object references instead of directly invoking constructors

- A `public static factory` method:
  - Maintains a cache of objects for reuse
  - Creates new instances, if the cache is depleted
  - Produces an object that subclasses the method's return type, as shown in the following code snippet:

```
NumberFormat nf = NumberFormat.getInstance();
```

- To call static methods:
  - Use the class name
    - Avoid using an object reference, as shown in the following code snippet:

```
double d = Math.random();
StaticUtilityClass.printMessage();
StaticUtilityClass uc = new
StaticUtilityClass();
uc.printMessage();
sameClassMethod();
```

Here, the object reference will work but it is misleading.

- Static variables:
  - Can be accessed without creating an instance of a class
  - Also called class variables
  - Limited to a single copy per JVM
  - Useful for containing shared data
  - Initialized when the containing class is first loaded
  - Shared by all object instances
  - Store data for static methods
  - Cause class loading on being accessed

- The following code snippet shows how to use a static variable:

```
double p = Math.PI;
. . .
new StaticCounter();
new StaticCounter();
System.out.println("count: "+
StaticCounter.getCount());
```

Here, the `StaticCounter()` method is called twice. Therefore, the `counter` variable declared inside this method will be incremented twice.
Thus, the final value of the `counter` variable will be 2.

- If all the members of a class are static, then use a private constructor to prevent object instantiation.

- The `static import` statement makes the static members of a class available using their simple name.
- The following code snippet shows how to call the `Math.random()` method:

```
import static java.lang.Math.random;
import static java.lang.Math.*;
public class StaticImport
{
 public static void main(String[] args) {
 double d = random();
}
}
```

Here, the `random()` method is called without the fully-qualified name, `import static java.lang.Math.random;`.

- Final methods:
  - Declared using the `final` keyword
  - Cannot be overridden
- The following code snippet shows how to declare a final method:

```
public class MethodParentClass{
public final void printMessage() {
 System.out.println("This is a final method");
 }}
 public class MethodChildClass extends
 MethodParentClass
 {
  public void printMessage(){
     System.out.println("Cannot override method");
 }}
```

This will result in compilation error because the `printMessage()` method is declared as `final` in the base class.

- Final classes:
  - Can be declared using the `final` keyword
  - Cannot be extended, as shown in the following code snippet:

```
public final class FinalParentClass { }
// compile-time error
public class ChildClass extends FinalParentClass
{
}
```

The `FinalParentClass` class is declared as `final` and does support inheritance.

- Final variables:
  - Declared using the `final` modifier
  - May not change their values after they are initialized
  - Can be of the following types:
    - Class fields
    - Method parameters
    - Local variables
  - Help in bug prevention and thread safety.
- Final references:
  - Must always reference the same object
  - Allow the content of the object to be modified

- Final fields:
  - Should be assigned value during the declaration
  - That are also static, are considered as constants
- The constant fields conventionally use identifiers having only uppercase letters and underscores.