1. How to Create an Object in Java?

To create an object in Java, you use the `new` keyword followed by a call to a constructor. Here's an example:

```
java
Copy code
public class MyClass {
    // Class constructor
    MyClass() {
        System.out.println("Object created!");
    }

    // Class method
    void display() {
        System.out.println("Hello, World!");
    }

    public static void main(String[] args) {
        // Creating an object of MyClass
        MyClass obj = new MyClass();

        // Calling a method using the object
        obj.display();
    }
}
```

2. What is the use of the `new` keyword in Java?

The `new` keyword in Java is used to create new objects. It allocates memory for the new object and returns a reference to it. The `new` keyword is followed by a call to a constructor, which initializes the new object.

3. What are the different types of variables in Java?

In Java, there are three main types of variables:

1. **Local Variables**: These are declared inside a method, constructor, or block. They are created when the method, constructor, or block is entered and destroyed when it is exited.

   ```
   public void myMethod() {
       int localVar = 10; // Local variable
   }
   ```

2. **Instance Variables**: These are declared in a class but outside any method, constructor, or block. They are created when an object of the class is created and destroyed when the object is destroyed.

   ```
   public class MyClass {
       int instanceVar; // Instance variable
   }
   ```

3. **Class (Static) Variables**: These are declared with the `static` keyword in a class but outside any method, constructor, or block. There is only one copy of the static variable, and it is shared among all the instances of the class.

   ```
   public class MyClass {
       static int staticVar; // Class variable
   }
   ```

4. What is the difference between Instance variables and Local variables?

- **Scope**:
  - **Instance Variable**: The scope is the entire class, and they are accessible from any method, constructor, or block of the class.
  - **Local Variable**: The scope is limited to the method, constructor, or block in which they are declared.
- **Lifetime**:
  - **Instance Variable**: They exist as long as the object exists.
  - **Local Variable**: They exist only during the execution of the method, constructor, or block they are declared in.
- **Initialization**:
  - **Instance Variable**: They are initialized to default values if not explicitly initialized.
  - **Local Variable**: They must be explicitly initialized before use.

5. In which area memory is allocated for instance variable and local variable?

- **Instance Variable**: Memory for instance variables is allocated in the heap memory when the object is created.
- **Local Variable**: Memory for local variables is allocated on the stack when the method, constructor, or block is invoked.

6. What is method overloading?

Method overloading is a feature in Java that allows a class to have more than one method with the same name, as long as their parameter lists are different (different number of parameters, different types of parameters, or both). It is a way to achieve polymorphism. Here's an example:

```
public class MyClass {
    // Method with one parameter
    void display(int a) {
        System.out.println("Argument: " + a);
    }

    // Overloaded method with two parameters
    void display(int a, int b) {
        System.out.println("Arguments: " + a + ", " + b);
    }

    public static void main(String[] args) {
        MyClass obj = new MyClass();

        // Calling the first display method
```

```
        obj.display(10);

        // Calling the overloaded display method
        obj.display(10, 20);
    }
}
```

In this example, the `display` method is overloaded with different parameter lists.