*Part2:

1.When the program runs, the following steps occur:

Memory Map:

Static Heap:

- Class definitions, including static variables

Stack:

- Method calls, local variables, and references to objects

Dynamic Heap:

- Objects created at runtime, including instance variables
- Program runs:

+ Step 1: Create a new instance of a class BeeColony and run its method.

+ Step 2: Create a new instance of a class FPTUniversity and run its method.

+ Step 3: Create a new instance of a class BeeColony and run its method.

+ Step 4: Create a new instance of a class FPTUniversity and run its method.

2.What is stored in the static heap, stack, dynamic heap?

- Static Heap: It stores class definitions, including static variables. In this program, class definitions for Organization, Colony, University, BeeColony, FPTUniversity, and Tester will be stored in the static heap.

- Stack: It stores method calls, local variables, and references to objects. In this program, the stack will store the method calls and local variables during the execution of the main() method in the Tester class.

- Dynamic Heap: It stores objects created at runtime, including instance variables. In this program, objects of classes Colony, BeeColony, University, and FPTUniversity will be created and stored in the dynamic heap.

3.Why is the Organization class abstract?

- The Organization class is declared as abstract because it contains an abstract method, communicateByTool(). An abstract class cannot be instantiated, meaning objects cannot be created directly from an abstract class. It serves as a base class for its subclasses (Colony and University) and provides common behavior and fields. However, since the communication method is specific to each subclass, it is marked as abstract, indicating that the subclasses must provide their own implementation.

4. Why must the Colony/University class implement the communicateByTool() method?

- The Colony and University classes are subclasses of the Organization class, which declares the abstract method communicateByTool(). When a class extends an abstract class, it must provide implementations for all abstract methods defined in the superclass. Therefore, the Colony and University classes must implement the communicateByTool() method to fulfill this requirement.

5.Polymorphism feature in the code:

- Polymorphism is demonstrated in the code through the use of the Role interface. The Role interface is implemented by the BeeColony and FPTUniversity classes. By creating objects of the Role interface and assigning them to instances of the BeeColony and FPTUniversity classes, polymorphism allows us to treat these objects as instances of the Role interface and invoke the common createWorker() method defined in the interface. This provides flexibility and code reusability.

6.Difference between an interface and an abstract class:

- Interface: An interface defines a contract of methods that a class implementing the interface must implement. It specifies the behavior that a class should adhere to. Interfaces cannot have method implementations, only method signatures. A class can implement multiple interfaces, allowing for multiple inheritance of behavior. Interfaces are used to achieve abstraction and provide a way for unrelated classes to share common behavior.
- Abstract Class: An abstract class is a class that cannot be instantiated and is typically used as a base class for its subclasses. It can contain both abstract and non-abstract methods. Abstract methods are declared without implementations and must be implemented by subclasses. Abstract classes can have instance variables and can provide common behavior to its subclasses. Unlike interfaces, a class can only extend a single abstract class, limiting the inheritance to a single base class.