1. **Questions:**
2. What is polymorphism?
3. What is a **polymorphic reference**?
4. How does **inheritance** relate to **polymorphism** in Java?
5. Can a **parameter** to a method i.e. "catch(Animal a)" be **polymorphic**?
6. Can a **reference** " Animal a " **only** point to **an object** created by instantiating the **Animal** class?
7. Can a **polymorphic reference invoke** a **method** that is **only declared** at the **object's class level**?
8. What is an **interface**?
9. What does it mean "public class A **implements** B"?
10. Can an **interface** declare **instance variables**?
11. How can you declare an **abstract method**?
12. What is the purpose of an **abstract method**?
13. Is it possible to **instantiate an interface**?
14. Are there any **differences** between **extending** a **class** and **implementing** an **interface**?
15. Can an **interface hierarchy** be used for **polymorphism**?
16. How does **interfaces** relate to **polymorphism** in Java?
17. Is **static binding** or **dynamic binding** more **efficient**?
18. Do we need **dynamic binding** for method **overloading** or **overriding**?
19. Do we need **dynamic binding** for **variable shadowing**?
20. What is the **difference** between **static** and **dynamic binding**?

1. Answer
2. Polymorphism is the ability of an object to take on many forms.
3. A polymorphic reference can point to multiple types of objects at different points in time
4. Inheritance is one process by which polymorphic references are created. A reference to a type higher in an inheritance hierarchy can always refer to an object of a type lower in an inheritance hierarchy due to the *is-a* relationship.
5. A method can accept a polymorphic reference; this may give the method more flexibility than it would otherwise have.
6. This reference may point to an object of any type that is compatible with Animal. In particular, it may point to any object that is an instance of a class that is a subclass of Animal.
7. In general, a polymorphic reference can only be used to invoke methods that are known to the class of the reference variable. In order to call a method that is declared at the object's class level, the reference variable must be cast to be of the object's type as part of the call.
8. An interface is mostly a collection of abstract methods
9. Class A accepts the contract of class B. Class A has to implement all methods of class B
10. An interface may declare constants, but it may not declare instance variables.
11. An *abstract method* is a method header without a method body
12. The purpose of an abstract method is, that each class which extends / implements the superclass / interface has to implement this method
13. No, an interface cannot be instantiated, because it only contains abstract methods.
14. Yes. First, extending a class uses the extends keyword, while implementing an interface uses the implements keyword. More importantly, a class can implement multiple interfaces, while it can only extend a single class.
15. Yes. A reference variable can use an interface as its type, and can then refer to objects of classes that implement the interface. In this way, polymorphism works the same as it does with an inheritance hierarchy.
16. Interface is one process by which polymorphic references are created. A reference to a type higher in an interface hierarchy can always refer to an object of a type lower in an interface hierarchy due to the *is-a* relationship.
17. Late binding is less efficient than compile-time binding due to the overhead associated with determining the code that should be executed at run time.
18. Method overriding is responsible for dynamic binding, because the compiler cannot resolve it at compile time. The actual object having a method invoked on it could possible be one of several.
19. private, final and static methods and variables use static binding
20. Static binding in Java occurs during compile time while dynamic binding occurs during runtime. Static binding uses Type (class in Java) information for binding while dynamic binding uses object to resolve binding.