How do you decide if A should inherit B?

- Is it obvious that an A *is-a* B?
- Will clients want to use an A as a B?

How are method invocations on variables and expressions bound?

Dynamically, meaning method implementation is determined at runtime based on the class of the object, not the type of the variable or expression.

What do combinators do?

They compose operators and combine elements of some domain into new elements.

What is override modifier required for?

What is it optional for?

It is required for all members (*i.e.*, it must also be applied to fields).

It is optional if you're implementing an abstract member.

What do all constructors in Scala ultimately invoke?

How is this enforced?

They invoke the primary constructor.

Each non-auxiliary constructor must call another constructor as its first line?

Classes must be abstract if they have what?

Abstract members.

Prevent type derivation.

As in Java, with `final` keyword on class or method.

You can also use `sealed`, which only allows derivation in the same source file.

Give an alternative to inheritance, and when it should be used.

Composition is preferred if only code re-use is desired.

When should parameterless methods used?

When the function is pure, i.e. has no side effects and does not depend upon mutable state.

What do parametric field definitions allow?

A field defined as a class parameter allows visibility modifiers, `val` or `var`, and `override`.

```
class ArrayElement(
  val contents : Array[String]
) extends element
```

What does a subclass *not* inherit?

Private members and overriden members.

Parameterless methods vs. empty-paren methods

```
def width() : Int
def width : Int
```

If defined parameterless, it must be called that way.

This is to support the uniform access principle.

What's a problem with fragile base class?

How does Scala minimize it?

It causes inadvertent breakage of subclasses by changing a superclass.

Forcing override notation helps solve it.

Name the class namespaces in Java and Scala.

- Java: fields, methods, types, packages
- Scala: values (fields, methods, packages, singletons), types (class and trait names)

What does Scala's class namespace allow?

It allows the overriding of parameterless and empty-paren methods with a val.

Give the Scala two factory patterns.

Use companion object to provide methods for making instances of the companion class.

Then client code can import just the factory methods (which share the same name).

Optionally, make concrete subclasses private inside singleton to hide details. Only the visible superclass is returned.

Why do packages share the same namespace as fields and methods?

To allow importing packages themselves and not just the names of types.

This also allows the importing of fields and methods of singleton objects.

How does Scala achieve compatibility with Java's non-support of uniform access?

It allows empty-paren methods to override parameterless ones and vice-versa.