**Favour composition over inheritance is a one of the popular object oriented design principle, which helps to create flexible and maintainable code in object oriented languages.**

1) In case of extending a class, you only get facilities which are available at compile time.

e.g:

Class FileReader

{

Read()

{

// code to read from file

}

}

Class FileWriter

{

Write()

{

// code to write to file

}

}

Class MyApplication:FileWriter, FileReader

{

Perform()

{

Read() // of FileReader // tight coupling

Write() // of FileWriter // tight coupling

}

}

drawbacks:

* Tight coupling- if base class ( FileReader or FileWriter) is changed, sub class (MyApplication) will break.
* inheritance breaks encapsulation. **white-box reuse**. That is ,with inheritance, the parent class implementation is often visible to the subclasses.  
    
  2) Composition offers better test-ability of a class than Inheritance. If one class is composed of another class, you can easily create Mock Object representing composed class for the sake of testing. Inheritance doesn't provide this luxury. In order to test derived class, you must need its super class. Since unit testing is one of the most important thing to consider during software development, especially in test driven development, composition wins over inheritance.

e.g.

interface Reader

{

Void read();

}

Interface Writer

{

Void write();

}

Class FileReader implements Reader

{

Void read()

{

Code to read from file

}

}

Class FileWriter implements Writer

{

Void write()

{

Code to write to file

}

}

Class MyApplication

{

// program to interface, enables loose coupling

Reader ref1;

Writer ref2;

Public MyApplication(Reader ref1,Writer ref2)

{

This.ref1=ref1;

This.ref2=ref2;

}

Void perform()

{

Ref1.read(); // late binding

Ref2.write(); // late binding

}

}

In the above example “MyApplication” is composed of “Reader” and “Writer”.

Advantages:

* **black-box reuse** as it does not break encapsulation. MyApplication knows only selected functionalities from “Reader” and “Writer”.
* Loose coupling, program to interface. During runtime any implementations (such as “FileReader” or “SocketReader” and “FileWriter” or “SocketWriter”) can be passed to “Reader” or “Writer” respectively and “read()” method can be invoked on it polymorphically.  
    
  Most of the object oriented design patterns mentioned by Gang of Four *favour Composition over Inheritance*.

In short, don't use Inheritance just for the sake of code reuse, Composition allows more flexible and extensible mechanism to reuse the code.

Though there are certainly some cases where using Inheritance makes much sense like when a genuine parent child relation exists, but most of time it makes sense to favour composition over inheritance for code reuse.