1) R=
Herança é o mecanismo de orientação a objetos onde classes podem herdar métodos e atributos de outras classes, com o intuito de reaproveitar código da classe herdada, também chamada de super classe ou classe mãe.

2) R=
Uma superclasse é uma classe geralmente abstrada, que generaliza outros tipos de classe. Por exemplo, se em um sistema de simulação de um ecossistema temos a classe Animal, ela seria uma super classe genérica para representar todos os animais. Por outro lado, subclasses são classes especializadas, que herdam as caracteríticas das superclasses. No mesmo exemplo da simulação de um ecossistema, a classe Abelha seria uma subclasse da superclasse Animal. Se formos ainda mais longe, podemos criar uma classe AbelhaItaliana, que herda de Abelha, e que seria ainda mais especializada.

3) R=
No contexto de POO, significa dizer que existe uma classe chamada Pessoa, que é uma superclasse genérica para representar pessoas, enquanto Estudante é uma classe mais especializada, que certamente também é uma Pessoa, logo, uma subclasse de pessoa. Em um contexto de lógica simples, significa que Pessoa é um tipo genérico e geral, enquanto Estudante é um tipo mais específico. Todos os comportamentos que uma Pessoa tem, um Estudante também tem, mas nem tudo que é uma Pessoa precisa ter os mesmo comportamentos específicos de um Estudante.

4) R=
Herança permite o reaproveitamento de código. Por exemplo, se você tem uma classe Carro, e uma classe Avião, você sabe que ambos tem o comportamente de se abastecer. Então você pode criar uma superclasse MeioTransporte, implementar o código de abastecimento nela, e fazer Carro e Avião herdar dessa classe. Ambos os dois serão capazes de se abastecer, talvez precisem fazer alguns ajustes, mas ainda vão poder aproveitar o mesmo código.
Outra vantagem seria Polimorfismo. Em uma classe que simula o sistema de transporte, você pode declarar um carro e um avião, mas usar eles como se fossem apenas meios de transporte, tendo de descobrir ou tratá-los de forma mais específica apenas em determinados casos específicos. Embora esse meu exemplo não seja bom, Polimorfismo é uma outra vantagem que se pode ter da Herança.
Atributos privados só podem ser acessados pela própria classe, nem mesmo classes filhas tem acesso a eles. Isso é útil quando você quer proteger certos dados e comportamentos até mesmo de classes que vão herdar a sua classe. No entanto, caso isso seja um empecilho, é possível declarar esse atributo como protegido, com a palavra-chave protected (em Java). Essa palavra-chave faz com que o atributo continue sendo invisível para quem está fora da classe, mas visível e modificável para classes que herdam a sua classe.

5) R= Uma seta fechada e vazia é usada para representar herança. A seta sai da subclasse e vai para a superclasse, ou seja, ela sai da classe que vai herdar as características e vai para a classe que vai ter as características herdadas.

6) R=
Ela pode ser usada durante uma sobrescrita de método, para chama o método correspondente da classe mãe, e também no construtor, para chamar o construtor da classe mãe. Ou seja, a palava-chave super é em geral usada para uma classe filha se referir a classe mãe.

7) R=
Porque declarar os atributos comuns a todas as classes em cada uma das diferentes classes que representam diferentes tipos de professor significa não aproveitar da mais poderosa vantagem da herança e da orientação a objetos: o reaproveitamento de código.

8) R=
Criaria uma super classe abstrada Transporte, e faria TransporteAereo e TransporteTerrestre herdarem dela. Depois, faria Carro herdar de TransporteTerrestre e Avião e Helicóptero herdar de TransporteAereo. A herença iria estar presente na especialização de classe genérica Transporte para as classes mais específicas Carro, Avião e Helicóptero.

9) R=
Quando você não tenta chamar o construtor da superclasse, o compilador vai chamar o construtor padrão da classe mãe. Se o construtor padrão da superclasse tiver um ou mais argumentos, isso vai gerar um erro, pois o construtor que o compilador tentará chamar é um sem argumentos. No geral, não é uma boa prática não chamar o construtor da superclasse, pois é o construtor que inicializar os atributos, e normalmente você vai querer que seu objeto tenha os atributos herdados da superclasse já inicializados.