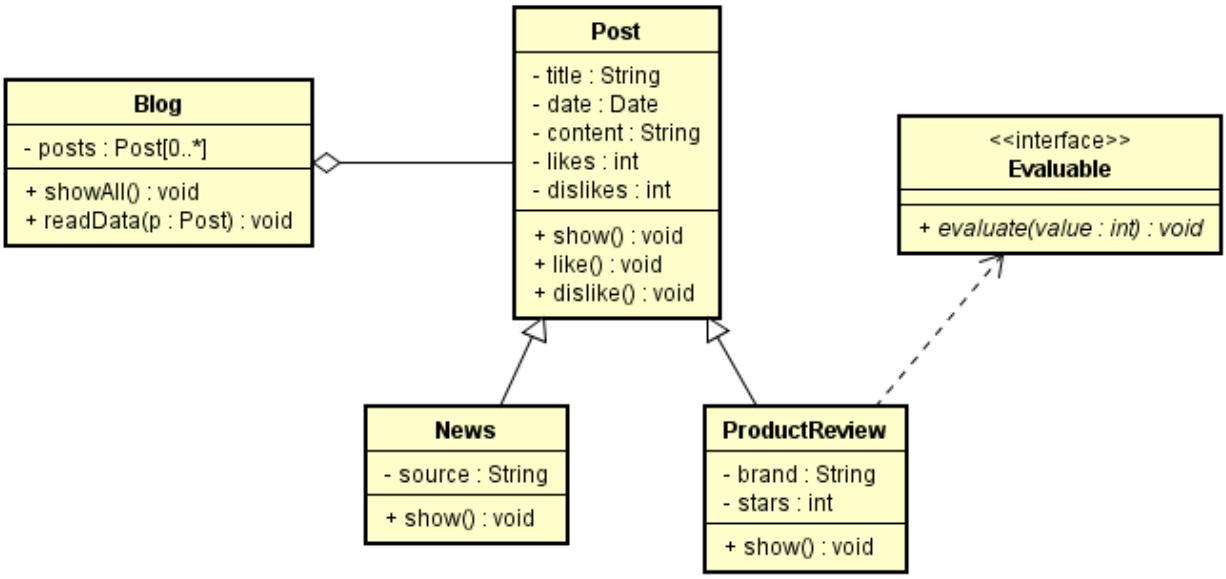
# Exercício 8 – Programação I

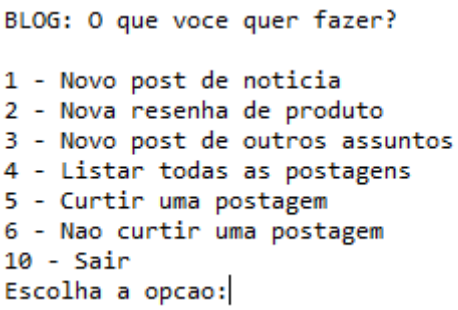
Assuntos abordados: **herança, polimorfismo, upcast, downcast, instanceof, interfaces**.

Você está desenvolvendo um blog sobre tecnologia. A estrutura de classes é exibida no diagrama abaixo:




Observe:

- **Blog** será o programa principal (e que em uma aplicação real seria substituído por uma interface web). Ele é um conjunto (agregação) de objetos **Post**. Esse conjunto deve ser implementado através de um **ArrayList** de objetos **Post**;
- O blog aceita posts sobre assuntos genéricos (classe **Post**) e posts mais específicos sobre notícias (classe **News**) ou resenha de produtos (**ProductReview**);
- Todos os posts podem ser curtidos (**like**) ou não curtidos (**dislike**);
- Todos os posts podem ser exibidos (**show**). Há uma exibição genérica com título, data, conteúdo, número de *likes* e de *dislikes*. As subclasses de **Post** devem estender a implementação de **show** e acrescentar suas informações específicas – fonte da notícia (**source**) ou marca do produto (**brand**) e número de estrelas (**stars**);
- As resenhas são avaliáveis, por isso, implementam a interface **Evaluable**. Esta interface contém um único método **evaluate**, que recebe como parâmetro um valor int. Na implementação deste método pela classe **ProductReview**, a avaliação será em escala de 1 a 10 estrelas, atribuídas pelo autor do post no momento de sua criação;
- Para as datas, utilize a classe **Date** do pacote **java.util**;
- O programa principal deve exibir o seguinte menu:




- As 3 primeiras opções criam os respectivos objetos (opção 3 deve criar um objeto Post) e armazenam em uma variável do tipo **Post** (aqui pode ocorrer *upcast*), depois a passam para o método **readData** da classe **Blog** e, em seguida, a armazenam no **ArrayList** de postagens;
- O método **readData** é chamado para ler os dados de uma nova postagem. Ele recebe um objeto do tipo **Post**, lê e atribui (usando *setters*) título, conteúdo e data (a data deve ser gerada automaticamente).
  - Se a postagem for um **News**, lê e atribui também a fonte da notícia (**source**).
  - Se a postagem for um **ProductReview**, lê e atribui também a marca (**brand**) e a avaliação (**stars**), chamando para registrar esta última o método **evaluate**.
  - Dicas: use o operador **instanceOf** para testar o tipo do objeto dentro de **readData**. Para chamar os métodos das subclasses, faça downcast no seguinte formato `((News) p).setSource(...);`
- A opção 4 chama o método **showAll**, o qual percorre o **ArrayList** exibindo o conteúdo de todas as postagens, invocando para isso o método **show** dos objetos **Post** ali armazenados (aqui ocorrerá polimorfismo!);
- ~~As opções 5 e 6 devem perguntar primeiramente o código do post (índice no array) a ser "curtido" ou "não curtido". Se o usuário tentar acessar um índice inexistente, exibir a mensagem "*Código inválido, postagem inexistente!*". Caso contrário, mostrar "*Postagem curtida!*" ou "*Postagem não curtida!*", conforme o caso;~~

Novos requisitos, referentes a **Exceções**:

- As opções 5 e 6 do menu devem perguntar primeiramente o código do post (índice no array) a ser "curtido" ou "não curtido". Deve haver tratamento de exceção para os seguintes casos:
  - se o usuário tentar acessar um índice inexistente, exibir a mensagem "Código inválido, postagem inexistente!". Caso contrário, mostrar "Postagem curtida!" ou "Postagem não curtida!", conforme o caso;
  - se o usuário digitar algo que não seja um número inteiro, exibir a mensagem "Entrada inválida! Utilize apenas números inteiros".
- O administrador do Blog decidiu que não devem ser permitidos 2 ou mais posts com título exatamente iguais. Para tanto, ao ler o título de um novo post no método readData, você deve fazer uma busca na lista de Posts a fim de verificar se aquele título já existe. Em caso afirmativo, lançar uma RuntimeException com uma mensagem. Não esqueça de capturar esta exceção no main, a cada chamada ao método readData.