Exercícios

Respostas enviadas em: 19/04/2021 20:13

1. Qual das alternativas abaixo descreve corretamente o objeto LiveData e a classe MutableLiveData?

A. Um objeto LiveData é parte integrante do ViewModel e implementa um padrão Factory. As classes MutableLiveData permitem que os objetos LiveData sejam definidos manualmente.

Por que esta resposta não é correta?
Um objeto LiveData é parte integrante do ViewModel e naturalmente implementa o padrão Observer. As classes MutableLiveData permitem que os objetos LiveData sejam definidos manualmente. Um objeto que implemente um padrão Observer não pode implementar um padrão Factory, uma vez que o padrão Factory seria uma fábrica de objetos, e não uma classe que "observa" alterações em estados de objetos criados para determinar ações a serem executadas. Além disso, é a classe MutableLiveData que permite alterar de forma manual um objeto LiveData.

B. Um objeto MutableLiveData é parte integrante do ViewModel e implementa um padrão Factory. As classes LiveData permitem que os objetos MutableLiveData sejam definidos manualmente.

Por que esta resposta não é correta?
Um objeto LiveData é parte integrante do ViewModel e naturalmente implementa o padrão Observer. As classes MutableLiveData permitem que os objetos LiveData sejam definidos manualmente. Um objeto que implemente um padrão Observer não pode implementar um padrão Factory, uma vez que o padrão Factory seria uma fábrica de objetos, e não uma classe que "observa" alterações em estados de objetos criados para determinar ações a serem executadas. Além disso, é a classe MutableLiveData que permite alterar de forma manual um objeto LiveData.

C. Um objeto MutableLiveData é parte integrante do ViewModel e naturalmente implementa o padrão Observer. As classes LiveData permitem que os objetos MutableLiveData sejam definidos manualmente.

Por que esta resposta não é correta?
Um objeto LiveData é parte integrante do ViewModel e naturalmente implementa o padrão Observer. As classes MutableLiveData permitem que os objetos LiveData sejam definidos manualmente. Um objeto que implemente um padrão Observer não pode implementar um padrão Factory, uma vez que o padrão Factory seria uma fábrica de objetos, e não uma classe que "observa" alterações em estados de objetos criados para determinar ações a serem executadas. Além disso, é a classe MutableLiveData que permite alterar de forma manual um objeto LiveData.

D. Um objeto LiveData é parte integrante do MutableLiveData e naturalmente implementa o padrão Observer. As classes MutableLiveData permitem que os objetos ViewModel sejam definidos manualmente.

Por que esta resposta não é correta?
Um objeto LiveData é parte integrante do ViewModel e naturalmente implementa o padrão Observer. As classes MutableLiveData permitem que os objetos LiveData sejam definidos manualmente. Um objeto que implemente um padrão Observer não pode implementar um padrão Factory, uma vez que o padrão Factory seria uma fábrica de objetos, e não uma classe que "observa" alterações em estados de objetos criados para determinar ações a serem executadas. Além disso, é a classe MutableLiveData que permite alterar de forma manual um objeto LiveData.

Você acertou!

E. Um objeto LiveData é parte integrante do ViewModel e naturalmente implementa o padrão Observer. As classes MutableLiveData permitem que os objetos LiveData sejam definidos manualmente.

Por que esta resposta é a correta?
Um objeto LiveData é parte integrante do ViewModel e naturalmente implementa o padrão Observer. As classes MutableLiveData permitem que os objetos LiveData sejam definidos manualmente. Um objeto que implemente um padrão Observer não pode implementar um padrão Factory, uma vez que o padrão Factory seria uma fábrica de objetos, e não uma classe que "observa" alterações em estados de objetos criados para determinar ações a serem executadas. Além disso, é a classe MutableLiveData que permite alterar de forma manual um objeto LiveData.

Exercícios

Respostas enviadas em: 19/04/2021 20:13

2. Com base no material estudado, marque abaixo a alternativa que apresenta as opções mais adequadas para o desenvolvimento de uma tela de cadastro para aplicações móveis.

A. Sempre que possível evitar telas de cadastro extensas. Entretanto, caso sejam necessárias, exigir que o usuário preencha todos os dados em uma única sequência de telas.

Por que esta resposta não é correta?
Sempre que se cria telas de cadastro para aplicativos móveis, deve-se evitar telas extensas, com grande quantidade de dados. Para os casos em que são necessárias muitas informações, deve-se optar pelo preenchimento gradativo, não condicionando a utilização do aplicativo ao preenchimento completo do cadastro. Dados que não são de extrema importância não devem ser obrigatórios. Apesar de ser comum observar telas com rolagens, o mais indicado ainda é a utilização de várias telas. Evite rolagens, pois o usuário pode se confundir. Lembre-se também de que uma aplicação deve exigir tantos dados quanto forem necessários para seu funcionamento correto e para operações futuras, mesmo que isso implique em vários campos a serem preenchidos, nesse caso, o melhor a fazer é saber como organizá-los. Contudo, lembre-se, somente exija que um usuário realize cadastro se realmente for necessário. Algumas aplicações não dependem de cadastro prévio, não precisam de segurança de dados, são simples e pontuais. Nesse caso, exigir cadastro é desnecessário.

Você acertou!

B. Evitar cadastros extensos com solicitação de grande quantidade de dados. Quando isto for necessário, o indicado é estimular o preenchimento gradativo, não impedindo a utilização da ferramenta caso o cadastro não esteja completo.

Por que esta resposta é a correta?
Sempre que se cria telas de cadastro para aplicativos móveis, deve-se evitar telas extensas, com grande quantidade de dados. Para os casos em que são necessárias muitas informações, deve-se optar pelo preenchimento gradativo, não condicionando a utilização do aplicativo ao preenchimento completo do cadastro. Dados que não são de extrema importância não devem ser obrigatórios. Apesar de ser comum observar telas com rolagens, o mais indicado ainda é a utilização de várias telas. Evite rolagens, pois o usuário pode se confundir. Lembre-se também de que uma aplicação deve exigir tantos dados quanto forem necessários para seu funcionamento correto e para operações futuras, mesmo que isso implique em vários campos a serem preenchidos, nesse caso, o melhor a fazer é saber como organizá-los. Contudo, lembre-se, somente exija que um usuário realize cadastro se realmente for necessário. Algumas aplicações não dependem de cadastro prévio, não precisam de segurança de dados, são simples e pontuais. Nesse caso, exigir cadastro é desnecessário.

C. Optar por telas de cadastro com rolagem, dessa forma, ainda que extensas, o usuário poderá preencher rapidamente todos os campos sem que se sinta entediado.

Por que esta resposta não é correta?
Sempre que se cria telas de cadastro para aplicativos móveis, deve-se evitar telas extensas, com grande quantidade de dados. Para os casos em que são necessárias muitas informações, deve-se optar pelo preenchimento gradativo, não condicionando a utilização do aplicativo ao preenchimento completo do cadastro. Dados que não são de extrema importância não devem ser obrigatórios. Apesar de ser comum observar telas com rolagens, o mais indicado ainda é a utilização de várias telas. Evite rolagens, pois o usuário pode se confundir. Lembre-se também de que uma aplicação deve exigir tantos dados quanto forem necessários para seu funcionamento correto e para operações futuras, mesmo que isso implique em vários campos a serem preenchidos, nesse caso, o melhor a fazer é saber como organizá-los. Contudo, lembre-se, somente exija que um usuário realize cadastro se realmente for necessário. Algumas aplicações não dependem de cadastro prévio, não precisam de segurança de dados, são simples e pontuais. Nesse caso, exigir cadastro é desnecessário.

D. Evite telas de cadastro extensas, prefira desenvolver aplicações nas quais não sejam exigidas muitas informações do usuário. Dessa forma, é possível garantir que o usuário utilize a aplicação por mais tempo.

Por que esta resposta não é correta?
Sempre que se cria telas de cadastro para aplicativos móveis, deve-se evitar telas extensas, com grande quantidade de dados. Para os casos em que são necessárias muitas informações, deve-se optar pelo preenchimento gradativo, não condicionando a utilização do aplicativo ao preenchimento completo do cadastro. Dados que não são de extrema importância não devem ser obrigatórios. Apesar de ser comum observar telas com rolagens, o mais indicado ainda é a utilização de várias telas. Evite rolagens, pois o usuário pode se confundir. Lembre-se também de que uma aplicação deve exigir tantos dados quanto forem necessários para seu funcionamento correto e para operações futuras, mesmo que isso implique em vários campos a serem preenchidos, nesse caso, o melhor a fazer é saber como organizá-los. Contudo, lembre-se, somente exija que um usuário realize cadastro se realmente for necessário. Algumas aplicações não dependem de cadastro prévio, não precisam de segurança de dados, são simples e pontuais. Nesse caso, exigir cadastro é desnecessário.

E. Sempre iniciar uma aplicação exigindo cadastro. Seja uma tela simples ou complexa, a exigência do cadastro garante que o usuário utilizará a aplicação por mais tempo.

Por que esta resposta não é correta?
Sempre que se cria telas de cadastro para aplicativos móveis, deve-se evitar telas extensas, com grande quantidade de dados. Para os casos em que são necessárias muitas informações, deve-se optar pelo preenchimento gradativo, não condicionando a utilização do aplicativo ao preenchimento completo do cadastro. Dados que não são de extrema importância não devem ser obrigatórios. Apesar de ser comum observar telas com rolagens, o mais indicado ainda é a utilização de várias telas. Evite rolagens, pois o usuário pode se confundir. Lembre-se também de que uma aplicação deve exigir tantos dados quanto forem necessários para seu funcionamento correto e para operações futuras, mesmo que isso implique em vários campos a serem preenchidos, nesse caso, o melhor a fazer é saber como organizá-los. Contudo, lembre-se, somente exija que um usuário realize cadastro se realmente for necessário. Algumas aplicações não dependem de cadastro prévio, não precisam de segurança de dados, são simples e pontuais. Nesse caso, exigir cadastro é desnecessário.

Exercícios

Respostas enviadas em: 19/04/2021 20:13

3. Em diversos casos, como desenvolvedor, você terá que utilizar componentes do tipo RadioGroup e RadioButton. Caso tenha que fazê-lo, qual seria a alternativa correta para recuperação do RadioButton marcado na interface gráfica com o usuário?

A. Integer sVip = rgVip.setCheckedRadioButtonId();
Button rbVip = (RadioButton)findViewById(sVip);

Por que esta resposta não é correta?
Integer sVip = rgVip.getCheckedRadioButtonId();
Button rbVip = (RadioButton)findViewById(sVip);
Para recuperar a identificação do botão marcado em um RadioGroup, é preciso recorrer ao método **getCheckedRadioButtonId()** do objeto que representa o RadioGroup. Observe que não existe uma função CheckedRadioButtonId() ou somente getCheckedRadioButton(), veja também que os métodos iniciados com **set** tem o objetivo de atribuir valores e não recuperar. O segundo passo é recuperar a instância do objeto RadioButton referente ao ID que foi recuperado. O método é o **findViewById(idDoObjeto)**, em quase todas as respostas o método está correto, no entanto, seu antecessor não está.

B. Integer sVip = rgVip.CheckedRadioButtonId();
Button rbVip = (RadioButton)findViewById(sVip);

Por que esta resposta não é correta?
Integer sVip = rgVip.getCheckedRadioButtonId();
Button rbVip = (RadioButton)findViewById(sVip);
Para recuperar a identificação do botão marcado em um RadioGroup, é preciso recorrer ao método **getCheckedRadioButtonId()** do objeto que representa o RadioGroup. Observe que não existe uma função CheckedRadioButtonId() ou somente getCheckedRadioButton(), veja também que os métodos iniciados com **set** tem o objetivo de atribuir valores e não recuperar. O segundo passo é recuperar a instância do objeto RadioButton referente ao ID que foi recuperado. O método é o **findViewById(idDoObjeto)**, em quase todas as respostas o método está correto, no entanto, seu antecessor não está.

C. Integer sVip = rgVip.getCheckedRadioButtonId();
Button rbVip = (RadioButton)findViewById();

Por que esta resposta não é correta?
Integer sVip = rgVip.getCheckedRadioButtonId();
Button rbVip = (RadioButton)findViewById(sVip);
Para recuperar a identificação do botão marcado em um RadioGroup, é preciso recorrer ao método **getCheckedRadioButtonId()** do objeto que representa o RadioGroup. Observe que não existe uma função CheckedRadioButtonId() ou somente getCheckedRadioButton(), veja também que os métodos iniciados com **set** tem o objetivo de atribuir valores e não recuperar. O segundo passo é recuperar a instância do objeto RadioButton referente ao ID que foi recuperado. O método é o **findViewById(idDoObjeto)**, em quase todas as respostas o método está correto, no entanto, seu antecessor não está.

D. Integer sVip = rgVip.getCheckedRadioButton();
Button rbVip = (RadioButton)findViewById(sVip);

Por que esta resposta não é correta?
Integer sVip = rgVip.getCheckedRadioButtonId();
Button rbVip = (RadioButton)findViewById(sVip);
Para recuperar a identificação do botão marcado em um RadioGroup, é preciso recorrer ao método **getCheckedRadioButtonId()** do objeto que representa o RadioGroup. Observe que não existe uma função CheckedRadioButtonId() ou somente getCheckedRadioButton(), veja também que os métodos iniciados com **set** tem o objetivo de atribuir valores e não recuperar. O segundo passo é recuperar a instância do objeto RadioButton referente ao ID que foi recuperado. O método é o **findViewById(idDoObjeto)**, em quase todas as respostas o método está correto, no entanto, seu antecessor não está.

Você acertou!

E. Integer sVip = rgVip.getCheckedRadioButtonId();
Button rbVip = (RadioButton)findViewById(sVip);

Por que esta resposta é a correta?
Integer sVip = rgVip.getCheckedRadioButtonId();
Button rbVip = (RadioButton)findViewById(sVip);
Para recuperar a identificação do botão marcado em um RadioGroup, é preciso recorrer ao método **getCheckedRadioButtonId()** do objeto que representa o RadioGroup. Observe que não existe uma função CheckedRadioButtonId() ou somente getCheckedRadioButton(), veja também que os métodos iniciados com **set** tem o objetivo de atribuir valores e não recuperar. O segundo passo é recuperar a instância do objeto RadioButton referente ao ID que foi recuperado. O método é o **findViewById(idDoObjeto)**, em quase todas as respostas o método está correto, no entanto, seu antecessor não está.

Exercícios

Respostas enviadas em: 19/04/2021 20:13

4. Com base no que foi estudado no material, qual alternativa melhor se adequa aos conceitos, descrições e funcionalidades da classe ViewModel?

A. Uma classe ViewModel permite que os dados seja lidos e armazenados de forma permanente na memória secundária do dispositivo, sem que seja necessário qualquer outro recurso.

Por que esta resposta não é correta?
Uma classe ViewModel permite a persistência de dados durante todo o ciclo de vida de uma aplicação. Por meio dessa classe, é possível transferir dados entre as diferentes atividades de uma aplicação, além de criar ações de recuperação de dados em caso de falhas no sistema, o que sugere um armazenamento temporário dos dados. Um armazenamento permanente se daria pelo armazenamento de arquivos no dispositivo ou pelo uso de algum banco de dados e esta não é uma utilidade da classe ViewModel. Classes ViewModel não possuem qualquer relação com o banco de dados, a não ser manter os dados que por ventura tenham sido recuperados, temporariamente armazenados em memória, enquanto a aplicação está em execução.
Um padrão Dont Talk to Stranger, representa a não comunicação entre classes "estranhas" entre si. Ou seja, prega que classes de contextos diferentes não devem ter qualquer tipo de conexão entre si, o que não representa nenhum padrão da classe ViewModel.
Um padrão Abstract Factory é aplicado quando existe a necessidade de se criar diversos objetos dos mais variados tipos, sendo possível sua criação graças às interfaces que implementa. Trata-se de outro padrão que não é observado na classe ViewModel.

B. Uma classe ViewModel é aplicada, sobretudo, para abertura e manutenção de conexões com bases de dados, observando constantemente o estado dos dados na base.

Por que esta resposta não é correta?
Uma classe ViewModel permite a persistência de dados durante todo o ciclo de vida de uma aplicação. Por meio dessa classe, é possível transferir dados entre as diferentes atividades de uma aplicação, além de criar ações de recuperação de dados em caso de falhas no sistema, o que sugere um armazenamento temporário dos dados. Um armazenamento permanente se daria pelo armazenamento de arquivos no dispositivo ou pelo uso de algum banco de dados e esta não é uma utilidade da classe ViewModel. Classes ViewModel não possuem qualquer relação com o banco de dados, a não ser manter os dados que por ventura tenham sido recuperados, temporariamente armazenados em memória, enquanto a aplicação está em execução.
Um padrão Dont Talk to Stranger, representa a não comunicação entre classes "estranhas" entre si. Ou seja, prega que classes de contextos diferentes não devem ter qualquer tipo de conexão entre si, o que não representa nenhum padrão da classe ViewModel.
Um padrão Abstract Factory é aplicado quando existe a necessidade de se criar diversos objetos dos mais variados tipos, sendo possível sua criação graças às interfaces que implementa. Trata-se de outro padrão que não é observado na classe ViewModel.

Você acertou!

C. Uma classe ViewModel permite a persistência de dados durante todo o ciclo de vida de uma aplicação. Por meio dessa classe, é possível transferir dados entre as diferentes atividades de uma aplicação, além de criar ações de recuperação de dados em caso de falhas no sistema.

Por que esta resposta é a correta?
Uma classe ViewModel permite a persistência de dados durante todo o ciclo de vida de uma aplicação. Por meio dessa classe, é possível transferir dados entre as diferentes atividades de uma aplicação, além de criar ações de recuperação de dados em caso de falhas no sistema, o que sugere um armazenamento temporário dos dados. Um armazenamento permanente se daria pelo armazenamento de arquivos no dispositivo ou pelo uso de algum banco de dados e esta não é uma utilidade da classe ViewModel. Classes ViewModel não possuem qualquer relação com o banco de dados, a não ser manter os dados que por ventura tenham sido recuperados, temporariamente armazenados em memória, enquanto a aplicação está em execução.
Um padrão Dont Talk to Stranger, representa a não comunicação entre classes "estranhas" entre si. Ou seja, prega que classes de contextos diferentes não devem ter qualquer tipo de conexão entre si, o que não representa nenhum padrão da classe ViewModel.
Um padrão Abstract Factory é aplicado quando existe a necessidade de se criar diversos objetos dos mais variados tipos, sendo possível sua criação graças às interfaces que implementa. Trata-se de outro padrão que não é observado na classe ViewModel.

D. Uma classe ViewModel permite a persistência de dados durante todo o ciclo de vida de uma aplicação. Estas classes trabalham principalmente implementando o padrão Dont Talk To Stranger.

Por que esta resposta não é correta?
Uma classe ViewModel permite a persistência de dados durante todo o ciclo de vida de uma aplicação. Por meio dessa classe, é possível transferir dados entre as diferentes atividades de uma aplicação, além de criar ações de recuperação de dados em caso de falhas no sistema, o que sugere um armazenamento temporário dos dados. Um armazenamento permanente se daria pelo armazenamento de arquivos no dispositivo ou pelo uso de algum banco de dados e esta não é uma utilidade da classe ViewModel. Classes ViewModel não possuem qualquer relação com o banco de dados, a não ser manter os dados que por ventura tenham sido recuperados, temporariamente armazenados em memória, enquanto a aplicação está em execução.
Um padrão Dont Talk to Stranger, representa a não comunicação entre classes "estranhas" entre si. Ou seja, prega que classes de contextos diferentes não devem ter qualquer tipo de conexão entre si, o que não representa nenhum padrão da classe ViewModel.
Um padrão Abstract Factory é aplicado quando existe a necessidade de se criar diversos objetos dos mais variados tipos, sendo possível sua criação graças às interfaces que implementa. Trata-se de outro padrão que não é observado na classe ViewModel.

E. Uma classe ViewModel permite a persistência de dados durante todo o ciclo de vida de uma aplicação. Estas classes trabalham principalmente implementando o padrão Abstract Factory.

Por que esta resposta não é correta?
Uma classe ViewModel permite a persistência de dados durante todo o ciclo de vida de uma aplicação. Por meio dessa classe, é possível transferir dados entre as diferentes atividades de uma aplicação, além de criar ações de recuperação de dados em caso de falhas no sistema, o que sugere um armazenamento temporário dos dados. Um armazenamento permanente se daria pelo armazenamento de arquivos no dispositivo ou pelo uso de algum banco de dados e esta não é uma utilidade da classe ViewModel. Classes ViewModel não possuem qualquer relação com o banco de dados, a não ser manter os dados que por ventura tenham sido recuperados, temporariamente armazenados em memória, enquanto a aplicação está em execução.
Um padrão Dont Talk to Stranger, representa a não comunicação entre classes "estranhas" entre si. Ou seja, prega que classes de contextos diferentes não devem ter qualquer tipo de conexão entre si, o que não representa nenhum padrão da classe ViewModel.
Um padrão Abstract Factory é aplicado quando existe a necessidade de se criar diversos objetos dos mais variados tipos, sendo possível sua criação graças às interfaces que implementa. Trata-se de outro padrão que não é observado na classe ViewModel.

Exercícios

Respostas enviadas em: 19/04/2021 20:13

5. Em algumas ocasiões, a necessidade de preencher cadastros para ter acesso ao sistema/aplicativo pode ser suprimida, recuperando informações de outros sistemas ou redes sociais das quais o usuário participa. Qual das opções abaixo representa uma forma correta de implementar esta funcionalidade?

A. A funcionalidade pode ser implementada através de acordos firmados entre o desenvolvedor e a rede social da qual se deseja obter informações. Para isto, o desenvolvedor deverá entrar em contato com os responsáveis pelas redes sociais com as quais deseja interagir.

Por que esta resposta não é correta?
Desenvolvedores de aplicativos móveis podem utilizar dados disponibilizados pelos usuários em cadastros de redes sociais para preencher seus cadastros (ou parte deles). No caso dos desenvolvedores Android, esta coleta de dados é feita através do componente ***Google Sign in*** que mantém uma espécie de acordo "lógico" entre a aplicação e a rede social, permitindo a troca de informações.

Você acertou!

B. Um acordo "lógico" é realizado entre as partes, utilizando plug-ins ou códigos disponibilizados pelas proprietárias das redes sociais. Para o desenvolvedor Android, a comunicação pode ser feita através do componente/plug-in Google Sign in.

Por que esta resposta é a correta?
Desenvolvedores de aplicativos móveis podem utilizar dados disponibilizados pelos usuários em cadastros de redes sociais para preencher seus cadastros (ou parte deles). No caso dos desenvolvedores Android, esta coleta de dados é feita através do componente ***Google Sign in*** que mantém uma espécie de acordo "lógico" entre a aplicação e a rede social, permitindo a troca de informações.

C. Para redes sociais como Facebook e LinkedIn, o desenvolvedor terá que desenvolver a partir do "zero" todo o código para fornecer a funcionalidade, uma vez que não existem plug-ins que possam auxiliar na tarefa.

Por que esta resposta não é correta?
Desenvolvedores de aplicativos móveis podem utilizar dados disponibilizados pelos usuários em cadastros de redes sociais para preencher seus cadastros (ou parte deles). No caso dos desenvolvedores Android, esta coleta de dados é feita através do componente ***Google Sign in*** que mantém uma espécie de acordo "lógico" entre a aplicação e a rede social, permitindo a troca de informações.

D. Exceto para contas Google, não é possível utilizar o recurso descrito no enunciado da questão.

Por que esta resposta não é correta?
Desenvolvedores de aplicativos móveis podem utilizar dados disponibilizados pelos usuários em cadastros de redes sociais para preencher seus cadastros (ou parte deles). No caso dos desenvolvedores Android, esta coleta de dados é feita através do componente ***Google Sign in*** que mantém uma espécie de acordo "lógico" entre a aplicação e a rede social, permitindo a troca de informações.

E. Não existem recursos que possam auxiliar o desenvolvedor nesse tipo de atividade. Caberá somente a solicitação de uma autenticação em outra rede social e a execução da exportação dos dados para dentro do aplicativo desenvolvido.

Por que esta resposta não é correta?
Desenvolvedores de aplicativos móveis podem utilizar dados disponibilizados pelos usuários em cadastros de redes sociais para preencher seus cadastros (ou parte deles). No caso dos desenvolvedores Android, esta coleta de dados é feita através do componente ***Google Sign in*** que mantém uma espécie de acordo "lógico" entre a aplicação e a rede social, permitindo a troca de informações.