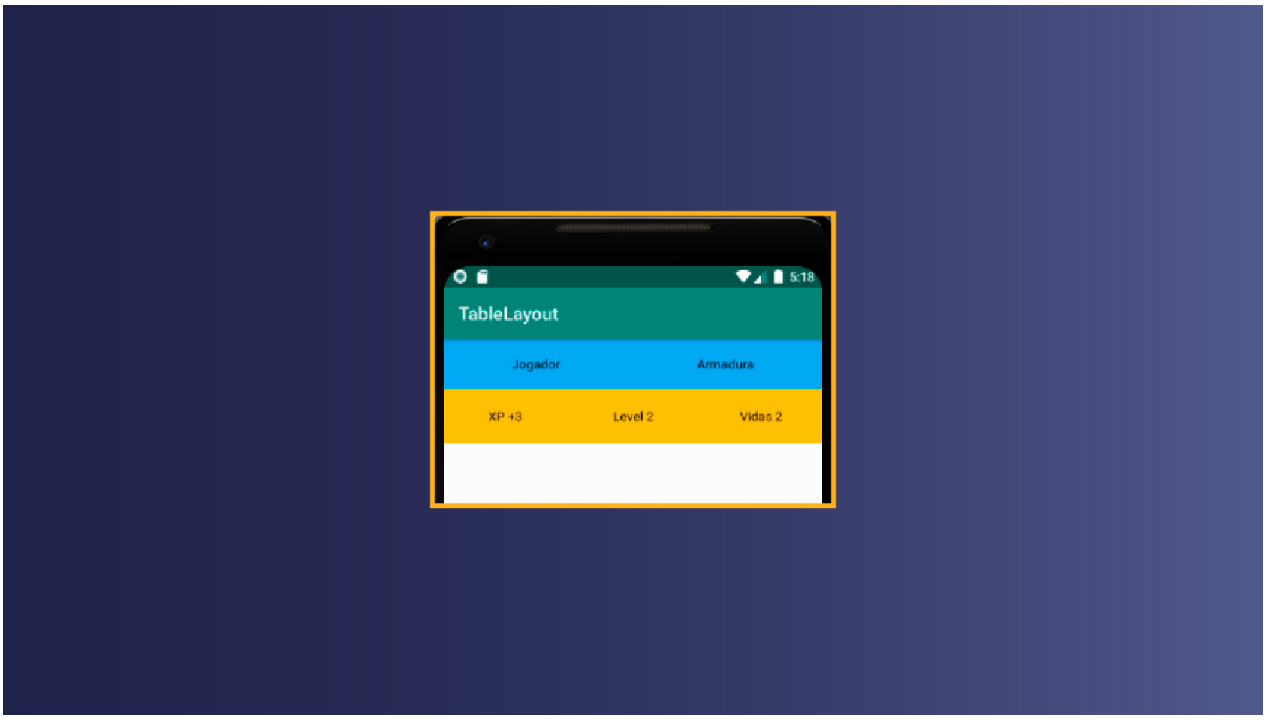
Exercícios

Respostas enviadas em: 19/04/2021 13:45

1. O TableLayout é uma especialização do LinearLayout, muito utilizado para criar formulários, telas de *login* ou para mostrar informações de forma tabular, como na imagem a seguir. Para ela, quantos <TableRow> são necessários?



Você não acertou!

- **A.** São necessários três TableRow.

  Por que esta resposta não é correta?
  O TableRow define a quantidade de linhas existentes em um TableLayout. Na imagem, há apenasduas linhas, então, caso você usasse apenas um TableRow não existiria uma segunda linha, já três, quatro e cinco TableRow criariam um leiaute com três, quatro e cinco linhas, respectivamente.

- **B.** São necessários cinco TableRow.

  Por que esta resposta não é correta?
  O TableRow define a quantidade de linhas existentes em um TableLayout. Na imagem, há apenasduas linhas, então, caso você usasse apenas um TableRow não existiria uma segunda linha, já três, quatro e cinco TableRow criariam um leiaute com três, quatro e cinco linhas, respectivamente.

Resposta correta

- **C.** São necessários dois TableRow.

  Por que esta resposta é a correta?
  O TableRow define a quantidade de linhas existentes em um TableLayout. Na imagem, há apenasduas linhas, então, caso você usasse apenas um TableRow não existiria uma segunda linha, já três, quatro e cinco TableRow criariam um leiaute com três, quatro e cinco linhas, respectivamente.

- **D.** É necessário um TableRow.

  Por que esta resposta não é correta?
  O TableRow define a quantidade de linhas existentes em um TableLayout. Na imagem, há apenasduas linhas, então, caso você usasse apenas um TableRow não existiria uma segunda linha, já três, quatro e cinco TableRow criariam um leiaute com três, quatro e cinco linhas, respectivamente.

- **E.** São necessários quatro TableRow.

  Por que esta resposta não é correta?
  O TableRow define a quantidade de linhas existentes em um TableLayout. Na imagem, há apenasduas linhas, então, caso você usasse apenas um TableRow não existiria uma segunda linha, já três, quatro e cinco TableRow criariam um leiaute com três, quatro e cinco linhas, respectivamente.

Anterior
Dica do Professor

Próximo
Na prática

Exercícios

Respostas enviadas em: 19/04/2021 13:45

2. Cada restrição ou *constraint* define a posição da *view* ao longo do eixo vertical ou horizontal. Ao usar o atributo app:layout_constraintStart_toStartOf="parent", informa-se ao leiaute que a *view* em questão vai

A. aparecer abaixo da *view parent.*

Por que esta resposta não é correta?
O atributo app:layout_constraintStart_toStartOf="parent" define que a *view* vai ser exibida ao lado de uma determinada *view*. Como o atributo setado é *parent*, ele vai se posicionar ao lado da ViewGroup em que ele se encontra. O atributo app:layout_constraintStart_toBottomOf aparece abaixo da *view*. Já o app:layout_constraintStart_toTopOf aparece acima.
O valor do atributo *parent* faz com que a *view* seja acorrentada ao "pai" da *view*, caso contrário seria necessário mencionar a *view* que você quer relacionar.

B. aparecer sobre a *view parent.*

Por que esta resposta não é correta?
O atributo app:layout_constraintStart_toStartOf="parent" define que a *view* vai ser exibida ao lado de uma determinada *view*. Como o atributo setado é *parent*, ele vai se posicionar ao lado da ViewGroup em que ele se encontra. O atributo app:layout_constraintStart_toBottomOf aparece abaixo da *view*. Já o app:layout_constraintStart_toTopOf aparece acima.
O valor do atributo *parent* faz com que a *view* seja acorrentada ao "pai" da *view*, caso contrário seria necessário mencionar a *view* que você quer relacionar.

Resposta correta

C. aparecer ao lado da *view parent.*

Por que esta resposta é a correta?
O atributo app:layout_constraintStart_toStartOf="parent" define que a *view* vai ser exibida ao lado de uma determinada *view*. Como o atributo setado é *parent*, ele vai se posicionar ao lado da ViewGroup em que ele se encontra. O atributo app:layout_constraintStart_toBottomOf aparece abaixo da *view*. Já o app:layout_constraintStart_toTopOf aparece acima.
O valor do atributo *parent* faz com que a *view* seja acorrentada ao "pai" da *view*, caso contrário seria necessário mencionar a *view* que você quer relacionar.

Você não acertou!

D. aparecer ao lado da imagem *view*, de nome Parent.

Por que esta resposta não é correta?
O atributo app:layout_constraintStart_toStartOf="parent" define que a *view* vai ser exibida ao lado de uma determinada *view*. Como o atributo setado é *parent*, ele vai se posicionar ao lado da ViewGroup em que ele se encontra. O atributo app:layout_constraintStart_toBottomOf aparece abaixo da *view*. Já o app:layout_constraintStart_toTopOf aparece acima.
O valor do atributo *parent* faz com que a *view* seja acorrentada ao "pai" da *view*, caso contrário seria necessário mencionar a *view* que você quer relacionar.

E. aparecer acima da imagem View Parent.

Por que esta resposta não é correta?
O atributo app:layout_constraintStart_toStartOf="parent" define que a *view* vai ser exibida ao lado de uma determinada *view*. Como o atributo setado é *parent*, ele vai se posicionar ao lado da ViewGroup em que ele se encontra. O atributo app:layout_constraintStart_toBottomOf aparece abaixo da *view*. Já o app:layout_constraintStart_toTopOf aparece acima.
O valor do atributo *parent* faz com que a *view* seja acorrentada ao "pai" da *view*, caso contrário seria necessário mencionar a *view* que você quer relacionar.

Exercícios

Respostas enviadas em: 19/04/2021 13:45

3. Para definir a posição de uma *view* no ConstraintLayout, deve-se ter, pelo menos, uma restrição horizontal e uma vertical para a *view*. Cada restrição representa uma conexão ou alinhamento para outra *view*, o leiaute "pai" ou uma diretriz invisível. Cada restrição ou *constraint* define a posição da *view* ao longo do eixo vertical ou horizontal. São formas de criação de restrições no ConstraintLayout os seguintes:

A. *Chains*, posicionamento circular, *android:shrinkColumns* e *layout_span*.

Por que esta resposta não é correta?
Atualmente, o ConstraintLayout aceita as seguintes restrições: posicionamento relativo, margens, posicionamento centralizado, posicionamento circular, comportamento de visibilidade, restrições de dimensão, *chains*, objetos do Virtual Helpers e otimizador. *Layout_span* e *shrinkColumns* são atributos de TableLayout.
As margens e os posicionamentos centralizados relacionam-se aos conteúdos das views em si, e não à sua posição.

Você acertou!

B. Margens, posicionamento centralizado, *chains* e posicionamento circular.

Por que esta resposta é a correta?
Atualmente, o ConstraintLayout aceita as seguintes restrições: posicionamento relativo, margens, posicionamento centralizado, posicionamento circular, comportamento de visibilidade, restrições de dimensão, *chains*, objetos do Virtual Helpers e otimizador. *Layout_span* e *shrinkColumns* são atributos de TableLayout.
As margens e os posicionamentos centralizados relacionam-se aos conteúdos das views em si, e não à sua posição.

C. TableLayout, posicionamento centralizado, *chains* e posicionamento circular.

Por que esta resposta não é correta?
Atualmente, o ConstraintLayout aceita as seguintes restrições: posicionamento relativo, margens, posicionamento centralizado, posicionamento circular, comportamento de visibilidade, restrições de dimensão, *chains*, objetos do Virtual Helpers e otimizador. *Layout_span* e *shrinkColumns* são atributos de TableLayout.
As margens e os posicionamentos centralizados relacionam-se aos conteúdos das views em si, e não à sua posição.

D. Margens,TableRow, *chains* e posicionamento circular.

Por que esta resposta não é correta?
Atualmente, o ConstraintLayout aceita as seguintes restrições: posicionamento relativo, margens, posicionamento centralizado, posicionamento circular, comportamento de visibilidade, restrições de dimensão, *chains*, objetos do Virtual Helpers e otimizador. *Layout_span* e *shrinkColumns* são atributos de TableLayout.
As margens e os posicionamentos centralizados relacionam-se aos conteúdos das views em si, e não à sua posição.

E. TableLayout, TableRow, *chains* e posicionamento circular.

Por que esta resposta não é correta?
Atualmente, o ConstraintLayout aceita as seguintes restrições: posicionamento relativo, margens, posicionamento centralizado, posicionamento circular, comportamento de visibilidade, restrições de dimensão, *chains*, objetos do Virtual Helpers e otimizador. *Layout_span* e *shrinkColumns* são atributos de TableLayout.
As margens e os posicionamentos centralizados relacionam-se aos conteúdos das views em si, e não à sua posição.

Exercícios

Respostas enviadas em: 19/04/2021 13:45

4. No desenvolvimento de uma tela para exibir dados em tabelados usa-se o TableLayout, o qual agrupa exibições em linhas. Ao utilizá-lo, os componentes são exibidos na tela conforme

A. a ordem em que foram escritos no leiaute, começando de baixo para cima.

Por que esta resposta não é correta?
O TableLayout exibe o leiaute conforme especificado no TableRow, sendo cada *view* uma coluna.
Os leiautes são montados de cima para baixo, conforme o *render* lê o arquivo xml e *constraint*, *chains* e outros atributos não interferem num TableLayout.

Você acertou!

B. a ordem em que foram escritos no leiaute apresentados na TableRow.

Por que esta resposta é a correta?
O TableLayout exibe o leiaute conforme especificado no TableRow, sendo cada *view* uma coluna.
Os leiautes são montados de cima para baixo, conforme o *render* lê o arquivo xml e *constraint*, *chains* e outros atributos não interferem num TableLayout.

C. a ordem em que foram escritos no leiaute apresentados na *chain*.

Por que esta resposta não é correta?
O TableLayout exibe o leiaute conforme especificado no TableRow, sendo cada *view* uma coluna.
Os leiautes são montados de cima para baixo, conforme o *render* lê o arquivo xml e *constraint*, *chains* e outros atributos não interferem num TableLayout.

D. a ordem em que foram escritos no leiaute, conforme definição do app:*layout*.

Por que esta resposta não é correta?
O TableLayout exibe o leiaute conforme especificado no TableRow, sendo cada *view* uma coluna.
Os leiautes são montados de cima para baixo, conforme o *render* lê o arquivo xml e *constraint*, *chains* e outros atributos não interferem num TableLayout.

E. a ordem em que foram escritos no leiaute, conforme a *constraint*.

Por que esta resposta não é correta?
O TableLayout exibe o leiaute conforme especificado no TableRow, sendo cada *view* uma coluna.
Os leiautes são montados de cima para baixo, conforme o *render* lê o arquivo xml e *constraint*, *chains* e outros atributos não interferem num TableLayout.

Exercícios

Respostas enviadas em: 19/04/2021 13:45

5. TableLayout é um ViewGroup que agrupa exibições em linhas, já o TableRow é responsável por criar as linhas na tabela e cada elemento adicionado a ele torna-se uma coluna. Caso seja necessário expandir uma coluna para ocupar mais de um espaço deve-se utilizar o atributo:

Você acertou!

A. *android:layout_span.*

Por que esta resposta é a correta?
Para expandir uma coluna no TableView é preciso atribuir o *android:layout_span* com o número de colunas desejadas.
- *shrinkColumns*: especifica em quais colunas se quer reduzir o espaço, quando necessário.
- *stretchColumns*: especifica em quais colunas se quer aumentar o espaço, quando necessário.

- TableRow: define uma nova linha na tabela.

- Layout_merge: atributo que não existe no *android.*

B. *android:shrinkColumns.*

Por que esta resposta não é correta?
Para expandir uma coluna no TableView é preciso atribuir o *android:layout_span* com o número de colunas desejadas.
- *shrinkColumns*: especifica em quais colunas se quer reduzir o espaço, quando necessário.
- *stretchColumns*: especifica em quais colunas se quer aumentar o espaço, quando necessário.

- TableRow: define uma nova linha na tabela.

- Layout_merge: atributo que não existe no *android.*

C. *android:stretchColumns.*

Por que esta resposta não é correta?
Para expandir uma coluna no TableView é preciso atribuir o *android:layout_span* com o número de colunas desejadas.
- *shrinkColumns*: especifica em quais colunas se quer reduzir o espaço, quando necessário.
- *stretchColumns*: especifica em quais colunas se quer aumentar o espaço, quando necessário.

- TableRow: define uma nova linha na tabela.

- Layout_merge: atributo que não existe no *android.*

D. *android:*TableRow.

Por que esta resposta não é correta?
Para expandir uma coluna no TableView é preciso atribuir o *android:layout_span* com o número de colunas desejadas.
- *shrinkColumns*: especifica em quais colunas se quer reduzir o espaço, quando necessário.
- *stretchColumns*: especifica em quais colunas se quer aumentar o espaço, quando necessário.

- TableRow: define uma nova linha na tabela.

- Layout_merge: atributo que não existe no *android.*

E. *android:layout_merge.*

Por que esta resposta não é correta?
Para expandir uma coluna no TableView é preciso atribuir o *android:layout_span* com o número de colunas desejadas.
- *shrinkColumns*: especifica em quais colunas se quer reduzir o espaço, quando necessário.
- *stretchColumns*: especifica em quais colunas se quer aumentar o espaço, quando necessário.

- TableRow: define uma nova linha na tabela.

- Layout_merge: atributo que não existe no *android.*