KeyTraining

Curs java pentru testare automata

JAVA

Cuprins curs:

- Access modifiers
- Polymorphism
- Method overloading
- Method overriding

KeyTraining

Acces modifiers

- Modificatorii de acces din java definesc limita pentru accesarea membrilor unei clase și a unei clase în sine.
- Cu alte cuvinte, modificatorii de acces sunt acei modificatori care sunt utilizați pentru a restricționa vizibilitatea (accesibilitatea) claselor, câmpurilor, metodelor sau constructorilor.
- Modificatorii de acces sunt, de asemenea, cunoscuți ca modificatori de vizibilitate.

KeyTraining

Private acces modifier

- Există următoarele aspecte despre modificatorul de acces privat pe care trebuie să îi aveți în vedere.
- 1. Modificatorul de acces privat în java se poate aplica unei variabile, metode, constructori, clase interioare, dar nu clasei exterioare care este clasa în sine.
- 2. Variabila de instanță poate fi privată, dar o variabilă locală nu poate fi privată.
- 3. Membrii privați (câmp, metodă sau constructor) ai unei clase nu pot fi accesați din afara clasei. Sunt accesibile numai în cadrul clasei.
- 4. Membrii privați ai unei superclase nu pot fi moșteniți în subclasă. Prin urmare, acestea nu sunt accesibile în subclasele.

KeyTraining

Private acces modifier

- 5. Dacă facem vreun constructor privat, nu putem crea un obiect al acelei clase dintr-o altă clasă și, de asemenea, nu putem crea subclasa acelei clase.
- 6. O clasă nu poate fi privată, cu excepția claselor interioare. Clasele interioare sunt membri ai clasei exterioare. Deci, membrii clasei pot fi privați.
- 7. Dacă declarăm o metodă ca fiind privată, aceasta se comportă ca o metodă declarată ca fiind definitivă. Nu putem apela metoda privată din afara clasei.

KeyTraining

Default acces modifier

- 1. Când modificatorul de acces nu este specificat membrilor unei clase sau a unei clase în sine, se numește modificator de acces implicit.
- 2. Valoarea implicită se poate aplica variabilei de instanță, variabilei locale, constructorului, metodelor, clasei interioare sau clasei externe.
- 3. Membrii impliciți ai unei clase sunt vizibili în interiorul clasei și oriunde în cadrul claselor din același pachet sau folder. Prin urmare, pot fi accesate din afara claselor din același pachet, dar nu pot fi accesate în afara pachetului.
- 4. Membrii impliciți pot fi moșteniți numai în subclasă în același pachet. Nu poate fi moștenit din afara pachetului.

KeyTraining

Protected acces modifier

- 1. Modificatorul de acces protected poate fi aplicat variabilelor de instanță, variabilelor locale, constructorilor, metodelor, claselor interioare, dar nu și clasei externe.
- 2. Membrii protected sunt accesibili în interiorul clasei și oriunde în cadrul claselor din același pachet și în afara pachetului, dar numai prin moștenire.
- 3. Membrii protected pot fi moșteniți în subclasă.
- 4. Dacă facem constructorul protejat atunci putem crea subclasa acelei clase în același pachet, dar nu în afara pachetului

KeyTraining

Public acces modifier

- 1. Modificatorul de acces public se poate aplica variabilelor de instanță, constructorilor, claselor interioare, clasei externe, metodelor, dar nu și cu variabilele locale.
- 2. Membrii publici ai unei clase pot fi folosiți oriunde.
- 3. Membrii publici ai unei clase pot fi moșteniți în orice subclasă.

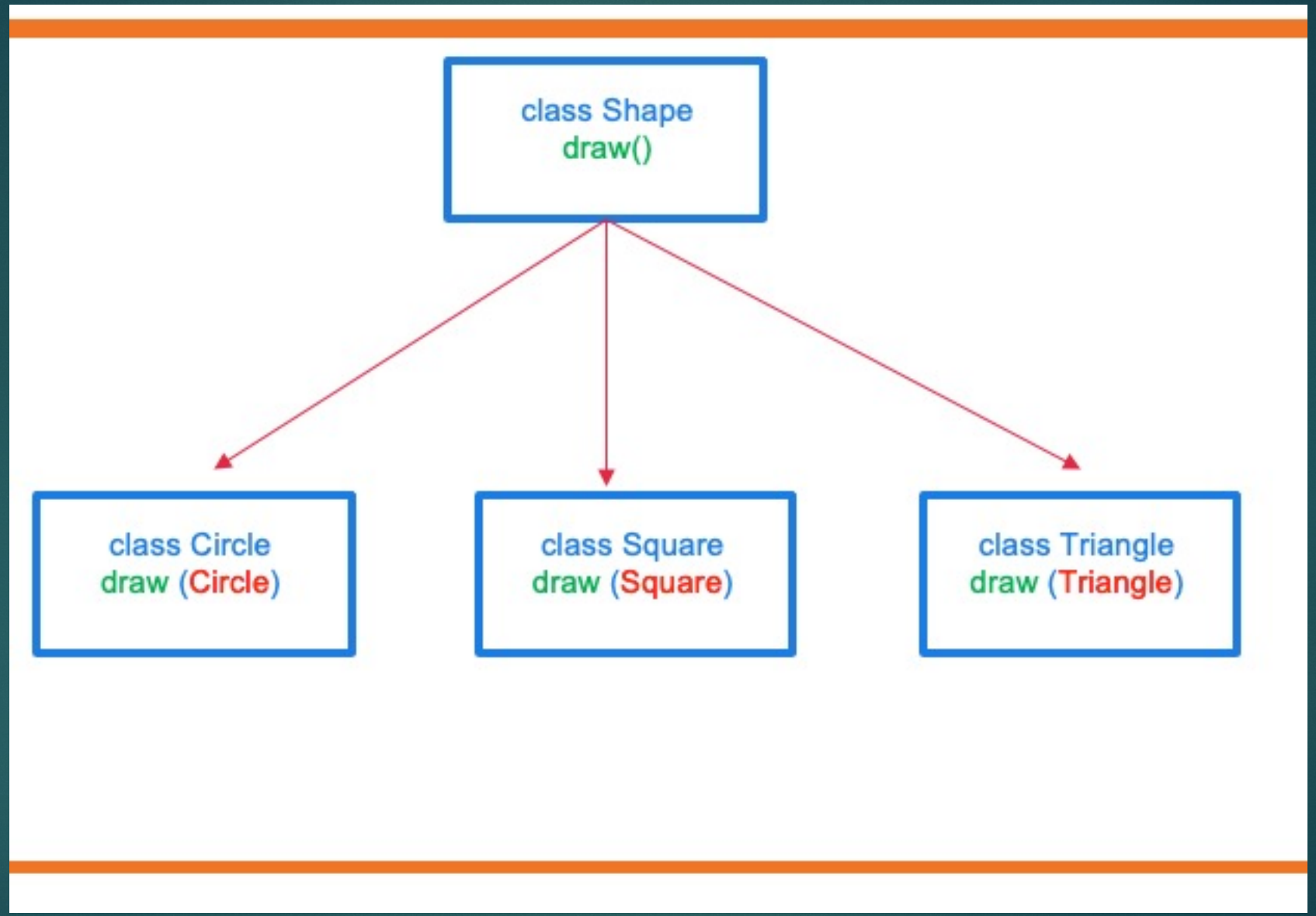
KeyTraining

acces modifier

Acces location	Public	Protected	Default	Private
Same class	yes	yes	yes	yes
Subclass in same package	yes	yes	yes	no
Other classes in same package	yes	yes	yes	no
Subclasses in other packages	yes	yes	no	no
Non-sublcasses on other packages	yes	no	no	no

OOP : Polymorphism

- Polimorfismul este unul dintre conceptele de bază ale limbajului de programare orientat pe obiecte.
- Cuvântul polimorfism este derivat din două cuvinte grecești: „poly” și ”morphs”.
- Cuvântul „ poly” implică multe și „ morphs” înseamnă forme.
- Prin urmare, polimorfismul înseamnă „multe forme”. Adică un lucru care poate lua mai multe forme.
- Polimorfismul este un concept prin care putem realiza o singură sarcină în moduri diferite. Adică, atunci când o singură entitate se comportă diferit în diferite cazuri, se numește polimorfism.


class Shape
draw()
class Circle
draw (Circle)
class Square
draw (Square)
class Triangle
draw (Triangle)

OOP : Polymorphism

- Practic, există două tipuri de polimorfism în Java :

- 1. Polimorfism static
- 2. Polimorfism dinamic

- În polimorfismul static, comportamentul unei metode este decis la compilare. Prin urmare, compilatorul Java leagă apelurile metodei cu definirea / corpul metodei în timpul compilării.
- Polimorfismul în timp de compilare poate fi realizat / implementat prin supraîncărcarea metodei (method overloading) în Java.

OOP : Polymorphism

- Un polimorfism care este prezentat în timpul rulării se numește polimorfism dinamic în Java.
- În polimorfismul dinamic, comportamentul unei metode este decis la rulare, prin urmare, JVM (Java Virtual Machine) leagă apelul metodei cu definiția metodei/corpul la runtime și invocă metoda relevantă în timpul runtime-ului când metoda este apelată.
- Acest lucru se întâmplă deoarece obiectele sunt create la runtime și metoda este numită folosind un obiect al clasei. Compilatorul Java nu are cunoștință de metoda care trebuie apelată la o instanță în timpul compilării. Prin urmare, JVM invocă metoda relevantă în timpul rulării.

OOP : Polymorphism

- Polimorfismul dinamic sau de rulare poate fi realizat / implementat în java folosind metoda de suprascriere (method overriding).
- Suprascrierea metodei este un mecanism în care o metodă a clasei de bază este suprascrisă în clasa derivată pentru a oferi o implementare mai specifică.

Method overloading

- Când o clasă are mai multe metode având același nume, dar cu liste de parametri diferiți, această caracteristică se numește supraîncărcare (method overloading) în Java.
- Cu alte cuvinte, atunci când mai multe metode sunt declarate cu același nume, dar cu semnături diferite ale metodelor, se spune că toate aceste metode sunt metode supraîncărcate.
- Acest proces este denumit supraîncărcare a metodei (method overloading).

Method overloading

- Compilatorul Java diferențiază metodele supraîncărcate prin semnăturile lor.
- Semnătura unei metode este definită de numele și lista de parametri.
- Tipul returnat al unei metode nu face parte din semnătura metodei. Nu joacă niciun rol în rezolvarea metodelor supraîncărcate.
- Când o clasă are mai multe metode având același nume cu lista de parametri diferiți, compilatorul diferențiază o metodă de o altă metodă prin verificarea numărului diferit de parametri sau a tipurilor de parametri trecuți. Prin urmare, ar trebui utilizat cu precauție, deoarece compilatorul Java nu ia în considerare tipul de returnare în timp ce diferențiază metoda supraîncărcată.

Method overloading

- Supraîncărcarea metodei este o caracteristică puternică în Java, dar ar trebui utilizată conform nevoilor.
- Ar trebui utilizat atunci când aveți nevoie de mai multe metode cu parametri diferiți, dar metodele fac același lucru.
- Dacă mai multe metode îndeplinesc sarcini diferite, nu utilizați conceptul de supraîncărcare a metodei.

Method overriding

- Suprascrierea (method overriding) în Java înseamnă redefinirea unei metode într-o subclasă pentru a înlocui funcționalitatea metodei din superclasa.
- Când metoda superclasei este suprascrisă în subclasă pentru a oferi o implementare mai specifică, se numește suprascriere (method overriding).