Betrachtet man nun das vorherige Beispiel der Spielerfigur (also der Klasse `Player`), läuft dieser Vorgang wie folgt ab. Die Klasse `Player` implementiert indirekt das Interface `Renderable`, da sie von der Klasse `Entity` abgeleitet ist. Nun wird während der Simulation durch den `EntityManager` die `draw()`-Methode von `Player` aufgerufen. Das `Player`-Objekt wird an den `MasterRenderer` übergeben. Dieser entnimmt einen `RenderedRenderable`-Behälter aus der Liste, der gerade nicht verwendeten Behälter (*Cache*) und befüllt diesen mit den Daten aus dem `Player`-Objekt. Jetzt können Änderungen im `Player` vorgenommen werden, ohne dass diese die Daten beeinflussen, die zum Zeichnen verwendet werden. Es wird also genau der Zustand dargestellt, den der `Player` hat, während die `draw()`-Methode aufgerufen wird. der mit den Daten aus `Player` befüllte `RenderedRenderable`-Behälter wird nun an den `OpaqueMasterRenderer` weitergegeben und in der Liste `usedRenderables` abgelegt.

> Betrachtet man nun das vorherige Beispiel der Spielerfigur (also der Klasse `Player`), läuft dieser Vorgang wie folgt ab. Die Klasse `Player` implementiert indirekt das Interface `Renderable`, da sie von der Klasse `Entity` abgeleitet ist. Nun wird während der Simulation durch den `EntityManager` die `draw()`-Methode von `Player` aufgerufen. Das `Player`-Objekt wird an den `MasterRenderer` übergeben. Dieser entnimmt einen `RenderedRenderable`-Behälter aus der Liste
>
> , der gerade nicht verwendeten Behälter (*Cache*) und befüllt diesen mit den Daten aus dem `Player`-Objekt. Jetzt können Änderungen im `Player` vorgenommen werden, ohne dass diese die Daten beeinflussen, die zum Zeichnen verwendet werden. Es wird also genau der Zustand dargestellt, den der `Player` hat, während die `draw()`-Methode aufgerufen wird. der mit den Daten aus `Player` befüllte `Rendered-Renderable`-Behälter wird nun an den `OpaqueMasterRenderer` weitergegeben und in der Liste `usedRenderables` abgelegt.

Betrachtet man nun das vorherige Beispiel der Spielerfigur (also der Klasse `Player`), läuft dieser Vorgang wie folgt ab. Die Klasse `Player` implementiert indirekt das Interface `Renderable`, da sie von der Klasse `Entity` abgeleitet ist. Nun wird während der Simulation durch den `EntityManager` die `draw()`-Methode von `Player` aufgerufen. Das `Player`-Objekt wird an den `MasterRenderer` übergeben. Dieser entnimmt einen `RenderedRenderable`-Behälter aus der Liste, der gerade nicht verwendeten Behälter (*Cache*) und befüllt diesen mit den Daten aus dem `Player`-Objekt. Jetzt können Änderungen im `Player` vorgenommen werden, ohne dass diese die Daten beeinflussen, die zum Zeichnen verwendet werden. Es wird also genau der Zustand dargestellt, den der `Player` hat, während die `draw()`-Methode aufgerufen wird. der mit den Daten aus `Player` befüllte `RenderedRenderable`-Behälter wird nun an den `OpaqueMasterRenderer` weitergegeben und in der Liste `usedRenderables` abgelegt.