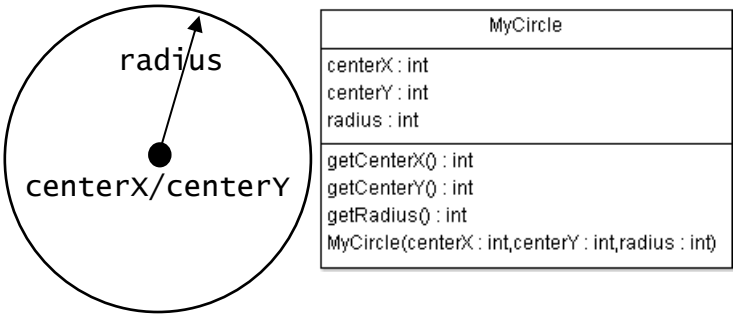
Informatik: Adapter und Proxy Pattern

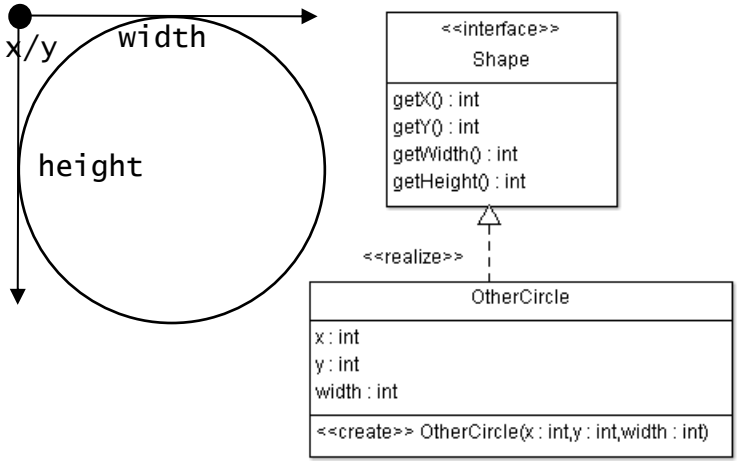
- Den Unterschied zw. einem Adapter und einem Proxy erkennen
- Verstehen welche Aufgaben ein Adapter zum Unterschied zu einem Proxy hat
- Unterschied zwischen einem Klassen- und Objekt-Adapter erkennen
- Einige Arten von Proxys kennen

Einleitendes Beispiel Adapter: `MyCircle`, `OtherCircle`

Position von meinen Objekten wird festgelegt von ihrem Mittelpunkt aus. So z.B. für den Kreis



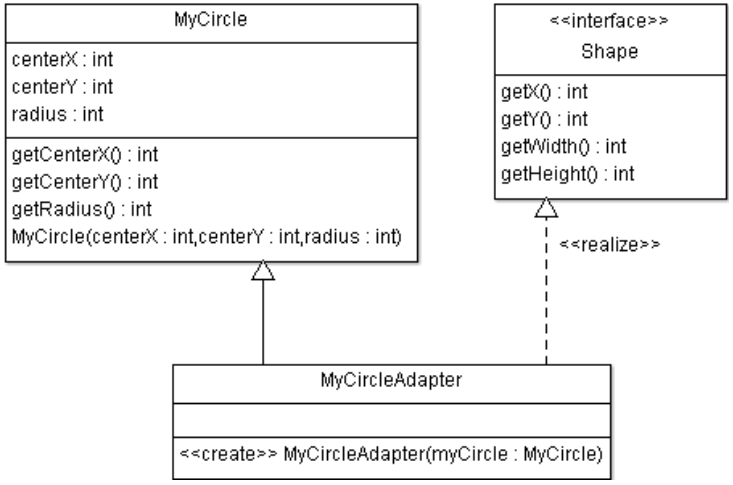
Andere Objekte bestimmen Position ausgehend von der linken oberen Ecke



PROBLEM: Meine Objekte sind mit den anderen nicht kompatibel

Gute Lösung: Klassen-Adapter

Um nicht die eigene Klasse umbauen zu müssen fungiert der Adapter als Übersetzer und macht meine Objekte kompatibel zu den anderen



AUFGABE

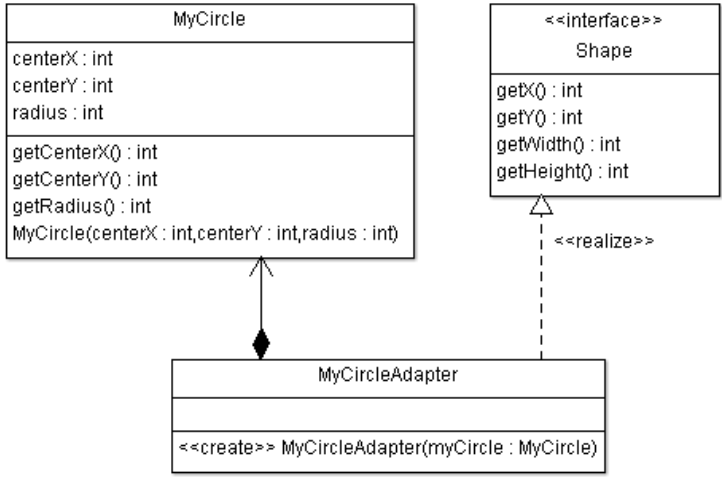
Implementieren Sie die Klasse `MyCircleAdapter`

☺ Objekt-Adapter kann bei Bedarf von einer anderen Klasse abgeleitet werden

☺ Anpassungsaufwand wird durch Adapter reduziert

Gute Lösung: Objekt-Adapter

Im Unterschied zum Klassen-Adapter wird beim Objekt-Adapter eine interne Member-Variable definiert über welche die Übersetzung läuft



AUFGABE

Implementieren Sie die Klasse `MyCircleAdapter`

```
MyCircleAdapter ma =
  new MyCircleAdapter(new MyCircle(20,20,10));
System.out.println(ma.getX());
System.out.println(ma.getY());
System.out.println(ma.getWidth());
```

```
<terminated>
10
10
20
```

HINWEIS: Objektadapter sind auch als *Hüllenklasse* oder *Wrapper-Klasse* bekannt

Adapter

- "Wird verwendet um die Schnittstelle einer Klasse in eine andere Schnittstelle zu übersetzen. Damit können Klassen zusammenarbeiten, die aufgrund inkompatibler Schnittstellen eigentlich nicht zusammenarbeiten können."

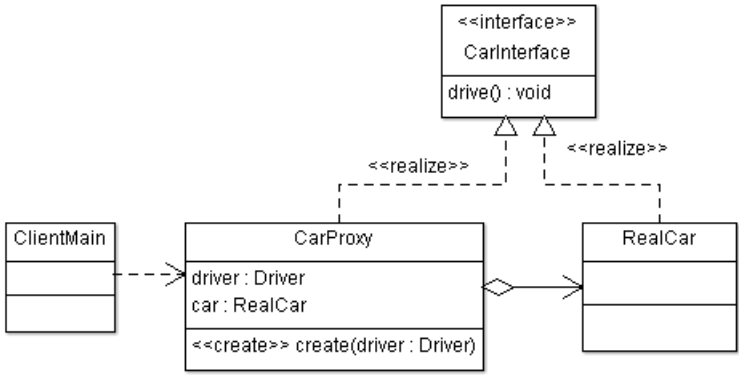
[1]

Unsere Klassenbezeichnung	Allgemeine Klassenbezeichnung
`Shape`	`TargetInterface`
`MyCircleAdapter`	`Adapter`
`MyCircle`	`Adaptee`

[1] Nach Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: "Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software". Addison-Wesley. 1. Auflage 1996 (*Gang of Four*)

Einleitendes Beispiel Proxy: Car

Es existiert das Interface `CarInterface` und die Klasse `RealCar`. Autos dürfen aber nur gefahren werden, wenn der Fahrer 18 Jahre alt ist



- Hauptprogramm `ClientMain` darf nicht direkt mit `RealCar` arbeiten
- `CarProxy` implementiert alle Methoden von `CarInterface` und wickelt diese nach der Fahrerkontrolle über `RealCar` ab
- Proxy kann noch zusätzliche Funktionalitäten auf sein Objekt definieren (z.B. um das Auto in der Waschanlage `CarWash` zu waschen)
- `RealCar`-Objekt in `CarProxy` wird nur bei Bedarf angelegt

```
public class CarProxy implements CarInterface
{
  protected Driver driver = null;
  protected CarInterface car = null;
  public CarProxy(Driver driver) {
    this.driver = driver;
  }
  public void drive() {
    if (driver.age >= 18) {
      if (car == null)
        car = new RealCar();
      car.drive();
    } else
      System.out.println("Sorry, driver is too young to drive");
  }
  public void wash(CarWash carWash) {
    if (car == null)
      car = new RealCar();
    carwash(car);
  }
}
```

Entstanden ist ein *Schutzproxy*, der den Zugriff auf das Objekt regelt

und ein *Virtueller Stellvertreterproxy*, der bei Bedarf erst das reale oft speicherintensive und komplexe Objekt erstellt

Es können auch *Remote Proxys* programmiert werden (siehe Übungen)

Proxy

- "Bietet einen Stellvertreter für ein anderes Objekt, um den Zugriff darauf zu steuern, den Ressourcenverbrauch zu senken und die Komplexität zu verringern."

[2]

Unsere Klassenbezeichnung	Allgemeine Klassenbezeichnung
`CarInterface`	`Subject`
`CarProxy`	`Proxy`
`RealCar`	`RealSubject`

[2] Nach Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: "Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software". Addison-Wesley. 1. Auflage 1996 (*Gang of Four*)