Fragen:

Wie kann diese Kommunikationsverbindung nun dennoch mit Hilfe einer zusätzlichen „Fabrik“-Klasse, welche die dazu notwendige Objekt-Erzeugung übernimmt, aufgebaut werden? In welchem Package sollte diese zusätzliche Klasse liegen?

Die Fabriklasse selber sollte eine Methode mit dem Rückgabetyp des Interfaces "Translator" enthalten, die Objekterzeugung übernimmt und diese Methode wird in der displaymethode der Clientklasse aufgerufen.
Die zusätzliche Fabrik-Klasse sollte in einem separaten Package liegen, das speziell für die Verwaltung oder Erzeugung von Objekten zuständig ist. Solche Klassen, die Objekte erstellen, gehören typischerweise in ein Package mit dem Namen factory. Außerdem besagt das Prinzip der Single Responsibility, dass jede Klasse oder Komponente nur eine Aufgabe haben sollte

Welches Entwurfsmuster könnte für die Problematik der Objekt-Erzeugung verwendet werden?

Für die Erzeugung von Objekten, ohne dass der Client explizit die Implementierung der zu erstellenden Klasse kennt, eignet sich das Factory-Methoden-Entwurfsmuster.
Ziel dieses Patterns ist die Problematik der Objekterzeugung zu lösen, indem es die Verantwortung zur Erzeugung von Objekten in eine spezielle Fabrikklasse auslagert. Der Client muss dann nicht mehr wissen, welche konkrete Klasse er instanziieren muss. Der Client ruft eine Methode z.B. createTranslator() in der Fabrik-Klasse auf, um eine Instanz zu erhalten. Der Client hat keine Kenntnis über die konkrete Klasse, die erstellt wird (z.B. GermanTranslator, sondern arbeitet nur mit einem Interface in diesem Fall Translator.

Wie muss man den Source Code des Interface ggf. anpassen, um mögliche Kompilierfehler zu beseitigen?

Um Kompilierfehler zu vermeiden, muss man das Interface so anpassen, dass es public wird.
Standardmäßig sind Interface-Methoden in Java public. Wenn eine Methode in der Implementierungsklasse GermanTranslator nicht als public deklariert wird, tritt ein Kompilierfehler auf, deswegen müssen wir das Interface und seine Methode öffentlich machen

Was ist der Vorteil einer separaten Test-Klasse?

Der Hauptcode bleibt übersichtlich, da die Testlogik nicht direkt im Produktionscode enthalten ist. Das erleichtert die Wartung und Lesbarkeit beider Teile. Außerdem ermöglicht eine separate Test-Klasse, alle Tests für eine bestimmte Klasse oder Funktionalität an einem Ort zu sammeln, was die Verwaltung der Tests vereinfacht.

Was ist bei einem Blackbox-Test der Sinn von Äquivalenzklassen?

Äquivalenzklassen helfen, die Anzahl der erforderlichen Tests zu reduzieren, ohne die Testabdeckung zu beeinträchtigen. Sie ermöglichen alle möglichen Eingabewerte zu testen, helfen Äquivalenzklassen, Eingabewerte in Gruppen aufzuteilen, bei denen jede Gruppe (Klasse) ähnliche Ergebnisse erzeugt. Anstatt jeden möglichen Wert einzeln zu testen, reicht es aus, je einen Repräsentanten pro Klasse zu testen.

Warum ist ein Blackbox-Test mit JUnit auf der Klasse Client nicht unmittelbar durchführbar?

Ein Blackbox-Test auf der Klasse Client ist nicht unmittelbar durchführbar, da die Klasse Client nur eine Verbraucherklasse ist und auf die Funktionalität von Translator zugreift, ohne eigene komplexe Logik zu implementieren. Die Methode display(int aNumber) verwendet eine Konsolenausgabe (System.out.println). Das bedeutet, dass der Test nicht auf den Rückgabewert der Methode zugreifen kann, sondern den Konsolenausgabe-Stream abfangen muss, was den Testaufwand erhöht. Zudem gibt die Methode display keinen Wert zurück, den man in einem Test direkt überprüfen kann. Ein Test müsste die Konsolenausgabe einfangen und überprüfen, was den Test schwieriger macht und nicht direkt mit JUnit getestet werden kann.