Parsons Problems — Week 10

Designing Data Types · Problem 1 of 2

Problem 1 — Drum

Rearrange the slips to form a complete Java program with an abstract `Instrument` class (holding an abstract `volume()` method and a concrete `printVolume()` method) and a `Drum` subclass that overrides `volume()`.

Example: `$ java Drum` prints `Volume: 8`

2 of the slips are distractors — they look plausible but do not belong in the solution.

```
Drum d = new Drum();
```

```
Instrument i = new Instrument();
```

```
abstract class Instrument {
```

```
abstract int volume();
```

```
d.printVolume();
```

```
int volume() { return 8; }
```

```
int volume();
```

```
public class Drum extends Instrument {
```

```
public static void main(String[] args) {
```

```
void printVolume() { System.out.println("Volume: " + volume()); }
```

```
}
```

```
}
```

```
}
```

Parsons Problems — Week 10

Designing Data Types · Problem 2 of 2

Problem 2 — Wizard

Rearrange the slips to form a complete Java program that declares two interfaces, `SpellCaster` and `Healer`, and a `Wizard` class that implements both of them.

Example: `$ java Wizard` produces:

```
Fireball damage: 40
Heal amount: 25
```

2 of the slips are distractors — they look plausible but do not belong in the solution.

```
System.out.println("Fireball damage: " + w.castFireball() + "\nHeal amount: " +
w.castHeal());
```

```
Wizard w = new Wizard();
```

```
interface Healer { int castHeal(); }
```

```
interface SpellCaster { int castFireball(); }
```

```
public class Wizard extends SpellCaster, Healer {
```

```
public class Wizard implements SpellCaster, Healer {
```

```
public int CastFireball() { return 40; }
```

```
public int castFireball() { return 40; }
```

```
public int castHeal() { return 25; }
```

```
public static void main(String[] args) {
```

```
}
```

```
}
```