In [ ]:
```
import sys
from pathlib import Path


PROJECT_ROOT = Path("..").resolve()
if str(PROJECT_ROOT) not in sys.path:
    sys.path.insert(0, str(PROJECT_ROOT))

print("PYTHONPATH OK:", PROJECT_ROOT)
```

```
PYTHONPATH OK: /home/adel/hpc_pour_ia
```

In [2]:
```
import matplotlib
print("backend:", matplotlib.get_backend())
%matplotlib inline
```

```
backend: module://matplotlib_inline.backend_inline
```

In [3]:
```
import matplotlib.pyplot as plt
from src.data import load_train_sample

X, y = load_train_sample(n_samples=16, img_size=(32, 32), seed=0)

plt.figure(figsize=(8, 8))
for i in range(16):
    plt.subplot(4, 4, i+1)
    plt.imshow(X[i])
    plt.title(f"Class {y[i]}")
    plt.axis("off")
plt.tight_layout()
plt.show()
```

```
2025-12-24 08:04:29.704064: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
2025-12-24 08:04:31.559452: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in performa
nce-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild Tens
orFlow with the appropriate compiler flags.
/home/adel/hpc_pour_ia/.venv/lib/python3.10/site-packages/numpy/core/getlimits.p
y:542: UserWarning: Signature b'\x00\xd0\xcc\xcc\xcc\xcc\xcc\xcc\xfb\xbf\x00\x00
\x00\x00\x00\x00' for <class 'numpy.longdouble'> does not match any known type: f
alling back to type probe function.
This warnings indicates broken support for the dtype!
  machar = _get_machar(dtype)
2025-12-24 08:05:11.893181: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
```

In [4]:
```
from pathlib import Path
import pandas as pd

# Racine du projet
PROJECT_ROOT = Path("..").resolve()
DATA_DIR = PROJECT_ROOT / "data"

meta = pd.read_csv(DATA_DIR / "Meta.csv")

print(meta.columns)
print(meta.head(5))
print("Nb classes meta:", meta.shape[0])
```

```
Index(['Path', 'ClassId', 'ShapeId', 'ColorId', 'SignId'], dtype='object')
          Path  ClassId  ShapeId  ColorId SignId
0  Meta/27.png       27        0        0   1.32
1   Meta/0.png        0        1        0   3.29
2   Meta/1.png        1        1        0   3.29
3  Meta/10.png       10        1        0   3.27
4  Meta/11.png       11        0        0   1.22
Nb classes meta: 43
```

In [5]:
```
import matplotlib.pyplot as plt
import pandas as pd
```

```
from pathlib import Path
from src.data import load_train_sample

PROJECT_ROOT = Path("..").resolve()
DATA_DIR = PROJECT_ROOT / "data"

meta = pd.read_csv(DATA_DIR / "Meta.csv").set_index("ClassId")

X, y = load_train_sample(n_samples=16, img_size=(32, 32), seed=0)

plt.figure(figsize=(10, 10))
for i in range(16):
    cid = int(y[i])
    row = meta.loc[cid]
    title = f"ID {cid} | shape {row['ShapeId']} | color {row['ColorId']}"
    plt.subplot(4, 4, i+1)
    plt.imshow(X[i])
    plt.title(title, fontsize=8)
    plt.axis("off")
plt.tight_layout()
plt.show()
```

In [ ]:

```
"""
ce script sépare le jeu de données d'entrainement (Train.csv) sur deux parties :
```

```
(1) un ensemble d'apprentissage et (2) un ensemble de validation,
ça c'est pour conserver la meme distribution des 43 classes dans les deux ensemb
"""
# --- Split train / validation --- =>>>>>>>

import pandas as pd
from pathlib import Path
from sklearn.model_selection import train_test_split

# chemins du projet
PROJECT_ROOT = Path("..").resolve()
DATA_DIR = PROJECT_ROOT / "data"

# charger les annotations
df = pd.read_csv(DATA_DIR / "Train.csv")

# séparation stratifiée (90% train / 10% validation)
train_df, val_df = train_test_split(
    df,
    test_size=0.10,
    random_state=42,
    stratify=df["ClassId"]
)

# vérifications
print("Train shape :", train_df.shape)
print("Validation shape :", val_df.shape)
print("Nb classes train :", train_df["ClassId"].nunique())
print("Nb classes validation :", val_df["ClassId"].nunique())

# vérifier que la distribution des classes est similaire
dist_train = train_df["ClassId"].value_counts(normalize=True).sort_index()
dist_val = val_df["ClassId"].value_counts(normalize=True).sort_index()
print("Différence maximale de distribution :", float((dist_train - dist_val).abs
```

```
Train shape : (35288, 8)
Validation shape : (3921, 8)
Nb classes train : 43
Nb classes validation : 43
Différence maximale de distribution : 2.7839561560730197e-05
```

In [8]:

```
"""
ce bloc sauvegarde les splits train/validation en CSVs pour rendre le projet rep
on réutilise exactement les memes données à chaque exécution.
"""
from pathlib import Path

PROJECT_ROOT = Path("..").resolve()
DATA_DIR = PROJECT_ROOT / "data"

train_path = DATA_DIR / "train_split.csv"
val_path = DATA_DIR / "val_split.csv"

train_df.to_csv(train_path, index=False)
val_df.to_csv(val_path, index=False)

print("Saved:", train_path)
print("Saved:", val_path)
```

```
Saved: /home/adel/hpc_pour_ia/data/train_split.csv
Saved: /home/adel/hpc_pour_ia/data/val_split.csv
```