Soluzione

Nel problema dobbiamo affrontare una parte strutturale (definire di cosa parliamo: le figure) ed una di controllo della grafica (inclusa la business logic del gioco e la gestione degli eventi).

Parte strutturale

Il gioco parla di vari tipi di Figura, di tipo cerchio e quadrato, e di tre diversi colori.
Le API ci offrono delle componenti grafiche (Circle e Rectangle) che per noi possono andare bene. Sono entrambe sottoclassi di Shape.
Decidiamo allora di costruire una classe in cui incapsulare la logica che ci serve, e che possa fornirci l'elemento grafico che vogliamo.
Useremo questa classe per controllare l'uguaglianza di sue istanze, e per ottenere la componente grafica di cui abbiamo bisogno.
Quindi la nostra classe Figura avrà

- una proprietà shape che rappresenta l'elemento grafico (uno Shape),
- una proprietà color che ne rappresenta il colore (un Color).

Nel suo costruttore decideremo in modo casuale quali valori assegnare a queste due proprietà, scegliendo tra i valori ammissibili.
Come generatore di numeri casuali usiamo `Math.random()` che ci restituisce un numero compreso nell'intervallo [0,1) (quindi 1 non compreso). Moltiplicando per n e prendendo la parte intera del risultato avremo quindi un numero intero compreso tra 0 e n-1.

Dobbiamo poi costruire il metodo equals, in due possibili modi.
Il primo si limita a confrontare se `this.color` e' uguale a `figura.color` (dove `figura` è il parametro della mia equals dopo averne fatto il cast): `color.equals(figura.color)`.

Il secondo era richiesto per la variante del gioco: oltre a verificare l'uguaglianza dei colori, deve controllare anche l'uguaglianza delle forme. Era segnalata la presenza di un tranello. Infatti potrei pensare di cavarmela dicendo `shape.equals(figura.shape)`, ma non funzionerebbe! Come mai?
Perché in questo caso attiverei la equals dell'oggetto che sto usando (di volta in volta, di Circle o di Rectangle a seconda di cosa trovo istanziato). Ora, come si può vedere dalle API, in Shape, Rectangle e Circle la equals non viene ridefinita, ma si eredita quella di Object!
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Rectangle.html
Pertanto due Shape (due Circle, due Rectangle…) sono uguali SOLO se sono identici, che non è quello che vogliamo!
Invece la equals di Color è ridefinita (si veda
https://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/Color.html).
Allora per sistemare le cose non dobbiamo controllare che le due istanze che confrontiamo abbiano lo stesso colore, ed *appartengano alla stessa classe*:

```
color.equals(figura.color) &&
      figura.shape.getClass().equals(shape.getClass())
```

Cosa ci resta da fare? Dobbiamo scrivere la hashcode. L'unica cosa a cui stare attenti è che la nostra funzione deve riflettere la logica della equals, quindi nel primo caso (solo uguaglianza di colore) passiamo solo la variabile color, nel secondo entrambe le variabili di istanza.
Quindi nel primo casto ritorneremo `Objects.hash(color);` mentre nel secondo `Objects.hash(shape,color);`

Come ultima cosa scriviamo un metodo toString per stampare una rappresentazione testuale degli oggetti della classe Figura.

Ecco il codice completo per la nostra classe:

Classe Figura

```
package it.unitn.disi.prog2.ronchet.myproject.myproject;

import javafx.scene.paint.Color;
```

```
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.scene.shape.Shape;

import java.util.Objects;

public class Figura {
    Shape shape=null;
    Color color;
    final Color colors[]={Color.BLUE, Color.RED, Color.YELLOW};
    Figura() {
        int colorIndex=(int)(Math.random()*3);
        color=colors[colorIndex];
        int shapeIndex=(int)(Math.random()*2);
        switch (shapeIndex) {
            case 0: shape=new Circle(30,30,30);
            break;
            case 1: shape=new Rectangle(60,60);
            break;
            default:
                System.out.println("Something has gone wrong...");
                System.exit(1);
        }
        shape.setFill(color);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Figura figura = (Figura) o;
        // soluzione per il testo standard:
        //return color.equals(figura.color);
        //soluzione per il testo con variante
        return color.equals(figura.color) &&
           figura.shape.getClass().equals(shape.getClass());
    }

    @Override
    public int hashCode() {
         // soluzione per il testo standard:
        //return Object.hash(color);
        return Objects.hash(shape, color);
    }

    @Override
    public String toString() {
        return "Figura{" +
                "shape=" + shape +
                ", color=" + color +
                '}';
    }
}
```

Parte logica

Questa sezione è molto più semplice. Dobbiamo innanzitutto identificare le componenti e decidere come disporle. Ci serve un Text nel quale scrivere i messaggi per l'utente, una zona dove mostrare le nostre figure, e due bottoni. Usiamo un Borderpane, e disponiamo in alto il Text, in basso i bottoni (all'interno di un HBox) e al centro le figure (anche questo un HBox).

Dobbiamo definire la logica del bottone Gioca:

- crea una nuova figura,
- aggiungila a box delle figure (e per comodità anche a una lista di Figure)
- se le figure sono due, disabilita il bottone Gioca, controlla se l'esito è vittoria o sconfitta, e comunicalo nel Text

Avremmo potuto fare a meno della lista di Figure, appoggiandoci direttamente alla lista dei Children dell'HBox che contiene le figure, ma in questo modo abbiamo una soluzione più pulita che separa gli aspetti logici (la nostra lista di Figure) da quelli grafici (la lista dei Children).

Ci resta da definire la logica del bottone Cancella, che dovrà:

- svuotare la lista di figure
- svuotare la lista dei Children
- ripristinare la scritta iniziale
- riprstinare il bottone Gioca (che potrebbe essere stato disabilitato, dipende da qual è il momento in cui premiamo "Cancella".

E' tutto, possiamo passare a vedere il codice.

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.text.Text;
import javafx.stage.Stage;

import java.util.ArrayList;

public class Gioco extends Application {
    @Override
    public void start(Stage stage)  {
        ArrayList<Figura> figures=new ArrayList<Figura>();
        BorderPane root=new BorderPane();
        // --------------------------------
        Text text=new Text("Gioca!");
        BorderPane.setAlignment(text, Pos.CENTER);
        // --------------------------------
        HBox boxFigure=new HBox();
        boxFigure.setAlignment(Pos.CENTER);
        BorderPane.setAlignment(boxFigure, Pos.BOTTOM_CENTER);
        Button giocaButton=new Button("Gioca!");
        class GiocaButtonHandler implements EventHandler<ActionEvent> {
            @Override
            public void handle(ActionEvent actionEvent) {
                // creiamo una nuova figura, aggiungiamola al box e alla
lista delle figure
                Figura f=new Figura();
                boxFigure.getChildren().add(f.shape);
                figures.add(f);
                int nfigures=figures.size();
                // se le figure sono più di una:
                // -- disabilitiamo il bottone Gioca
                // -- controlliamo se le figure sono uguali tra loro, e
diamo il messaggio opportuno
                if (nfigures>1) {
                    giocaButton.setDisable(true);
                    System.out.println(figures.get(0));
```

```
                    System.out.println(figures.get(1));

System.out.println(figures.get(0).color.equals(figures.get(1).color));

System.out.println(figures.get(0).shape.getClass().equals(figures.get(1).sh
ape.getClass()));

System.out.println(figures.get(0).equals(figures.get(1)));
                    if (figures.get(0).equals(figures.get(1))) {
                        text.setText("Hai vinto!");
                    } else {
                        text.setText("Hai perso!");
                    }
                }
            }
        };
        GiocaButtonHandler giocaButtonHandler=new GiocaButtonHandler();
        giocaButton.setOnAction(giocaButtonHandler);
        // ---------------------------------
        Button clearButton=new Button("Cancella");
        class ClearButtonHandler implements EventHandler<ActionEvent> {
            @Override
            public void handle(ActionEvent actionEvent) {
                boxFigure.getChildren().removeAll(boxFigure.getChildren());
                figures.removeAll(figures);
                giocaButton.setDisable(false);
                text.setText("Gioca!");
            }
        };
        ClearButtonHandler clearButtonHandler=new ClearButtonHandler();
        clearButton.setOnAction(clearButtonHandler);
        // ---------------------------------
        HBox boxBottoni=new HBox();
        boxBottoni.setAlignment(Pos.CENTER);
        BorderPane.setAlignment(boxBottoni, Pos.CENTER);
        boxBottoni.getChildren().addAll(giocaButton,clearButton);
        // ---------------------------------
        root.setTop(text);
        root.setCenter(boxFigure);
        root.setBottom(boxBottoni);
        // ---------------------------------
        Scene scene = new Scene(root, 320, 240);
        stage.setTitle("Gioco");
        stage.setScene(scene);
        stage.show();
    }

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